MASTER EN SCIENCES INFORMATIQUES

Implémentation d'un langage de coordination et application aux domaines du traitement de l'image et du son

Duysinx, Charles

Award date:
2009

Facultés Universitaires Notre-Dame de la Paix
Faculté d'Informatique

Implémentation d'un langage de coordination et application aux domaines du traitement de l'image et du son

Charles Duysinx

Mémoire présenté en vue de l'obtention du grade de Licencié en Informatique.
Année Académique 2008-2009

Table des matières

Table des figures

Liste des tableaux

Résumé

A l'heure des processeurs multi-coeurs, la parallélisation ou la distribution du code semblent pragmatiques, prometteuses et peut-être nécessaires. Les paradigmes sont nombreux. Pour notre part, nous avons choisi les langages de coordination et Linda en particulier. Linda est un langage de coordination qui sépare la coordination du calcul. Il permet donc à un langage hôte, grâce à un set de primitives, l'échange d'informations entre différents processus.

Pour implémenter Linda, nous avons choisi le langage Python qui possède l'avantage d'être reconnu par le monde scientifique comme un outil efficace pour le prototypage rapide d'applications.

Ce mémoire propose une application d'une implémentation personnelle de Linda à l'analyse de l'image et du son. Ces deux domaines ont été choisis parce qu'ils demandent encore aujourd'hui beaucoup de ressources.

Le mémoire débute par une introduction sur les concepts. Il présente ensuite une analyse de quatre implémentations existantes. Puis il propose une implémentation personnelle. Et enfin, il se termine par une analyse *in vivo* des outils qui ont été proposés dans la partie précédente.

– Mots-clés : Linda, Python, Pybrenda, Linuxtuples, Pylinda, NetworkSpaces, langage de coordination, traitement de l'image, traitement du son, calcul parrallèle, calcul distribué.

Abstract

At the time of multi-core processors, parallele computing, inter-process communication mecanisme or, distributed programming seems pragmatic, promising and perhaps necessary. The paradigms are numerous. We, for our part, have chosen the coordination languages and Linda in particular. Linda is a coordination language that separates coordination from computation. Linda allows a host language to exchange information between processes thanks to a set of primitives.

To design Linda, we chose Python, the qualities of which are well established in science. It is also recognized as an effective tools for rapid prototyping applications.

This report proposes an implementation of a customized application of Linda in the analysis of image and sound. These two areas were chosen because they still require significant resources.

After an introduction on the concepts, we analyze four implementations. We conclude with an analysis of *in vivo* the tools that have been presented in the preceding part.

– Keywords : Linda, Python, Pybrenda, Linuxtuples, Pylinda, NetworkSpaces, coordination languages, image processing, sound processing, parallel computing, distributed system, distributed program,distributed programming.

Remerciements

Je tiens à remercier toutes les personnes sans qui ce mémoire n'aurait pas pu être réalisé.

En premier lieu, le Professeur Jean-Marie Jacquet pour ses conseils et sa patience.

L'encadrement des cours du soir pour leur disponibilité.

Mes proches, Jean-françois, Laurence, Marie et en particulier Mylène, qui m'ont soutenu durant cette longue période, parfois difficile, en m'encourageant.

Antoine et Philippe, qui pendant ces études ont été pour moi, des condisciples précieux, des amis bienveillants et un soutien sans défaillance dans la poursuite et la réussite de mes études.

Introduction

A l'heure des processeurs multi-coeurs, la parallélisation ou la distribution du code semble pragmatique, prometteuse et peut-être de plus en plus nécessaire. En effet, n'a-t-on pas entendu ces derniers temps un responsable du principal fondeur de processeur déclarer que la course à la puissance monoprocesseur était derrière nous et que dorénavant, il fallait considérer la technologie multi-coeurs comme étant la réponse actuelle à notre recherche de puissance de calcul ?

L'arrivée du GPU et de ses centaines de processeurs comme auxiliaires de calcul a elle aussi un peu plus poussé la machine de Monsieur tout le monde et celle du chercheur vers plus de parallélisme.

Elle a également favorisé Internet et une profusion de nouvelles technologies dont le *Cloud Computing* pour n'en citer qu'une.

On le voit, les technologies qui, il y a peu, étaient encore réservées à une élite sont maintenant disponibles pour tout un chacun. Le calcul parallèle et le calcul distribué occupent tous les terrains.

Mais une technologie n'est rien sans des paradigmes efficaces pour l'exploiter.

En 1983, Carriero et Gelernter introduisent la notion d'espace de tuples (tuple spaces). D'abord en publiant *Linda in context* [Carriero et Gelernter, 1989], puis ensuite *Coordination Languages and their significance* [Gelernter et Carriero, 1992], ils démontrent que le modèle de coordination Linda qu'ils ont élaboré est simple et puissant pour seconder n'importe quel langage de programmation et lui permettre ainsi de devenir un langage parallèle.

Linda est un langage de coordination qui sépare la coordination du calcul. Linda permet donc à un langage hôte, grâce à un set de primitives, l'échange d'informations entre processus.

Pour implémenter Linda, nous avons choisi Python dont les qualités dans le domaine scientifique ne sont plus à démontrer. Parmi ses innombrables outils, citons Scipy, une bibliothèque soutenue par la communauté scientifique. Cette communauté qui est étendue et efficace organise depuis 2002 une conférence internationale, ce qui dénote un signe de maturité non seulement des développeurs mais aussi des scientifiques qui l'utilisent quotidiennement.

Parmi les qualités qu'offre un langage moderne et puissant, Python est aussi reconnu comme un outil efficace pour le prototypage rapide d'applications.

La première partie est consacrée aux concepts qui sont abordés dans ce mémoire.

Le traitement du son et de l'image trouve ses racines dans le traitement du signal. C'est pourquoi nous commencerons d'abord par explorer les bases du traitement du signal. Ensuite, nous aborderons les spécificités de l'image et du son d'un point de vue informatique : numérisation, stockage, format et concepts spécifiques au domaine seront étudiés.

Le modèle de coordination Linda sera ensuite décrit. Il nous est apparu important de remonter aux sources. En effet, nous voulions situer la naissance de ce paradigme et le replacer ainsi dans son contexte historique. Nous présenterons ensuite les primitives sous une angle contemporain qui tentera

de leur donner plus de rigueur scientifique sur le plan de la grammaire et de la sémantique.

Nous terminerons cette première partie par présenter brièvement le langage Python ainsi que les motivations qui nous ont poussés à le choisir pour explorer le domaine.

La deuxième partie, quant à elle, présente quatre implémentations de Linda en Python : Pybrenda, Linuxtuples, Pylinda et NetworkSpaces. Ces différentes implémentations seront analysées du point de vue de leur architecture logicielle, de leur implémentation des primitives du langage de coordination Linda, de leur gestion de l'espace de tuples et de leur distribution. Nous parlerons brièvement du quatrième mousquetaire Networksapce. Cette dernière implémentation, comme nous le verrons, n'a été portée à notre connaissance que très tardivement.

La troisième partie décrira la démarche que nous avons suivie pour proposer une implémentation de Linda compatible syntaxiquement avec Pylinda. Le cahier des charges plantera le décor. Puis, nous présenterons les choix de méthodologie, d'architecture et d'implémentation proprement dite de Linda. Un mode d'emploi de notre implémentation sera aussi proposé. Une comparaison avec Pylinda sera également faite en reprenant la grille d'analyse que nous aurons utilisée dans la deuxième partie.

La quatrième et dernière partie de notre mémoire sera une analyse *in vivo* des outils que nous proposons. Il nous a paru important de confronter implémentation au domaine du traitement de l'image et du son. Cette partie se veut avant tout génératrice de pistes de réflexions quant à l'utilisation pratique et à l'efficacité de notre API.

Première partie

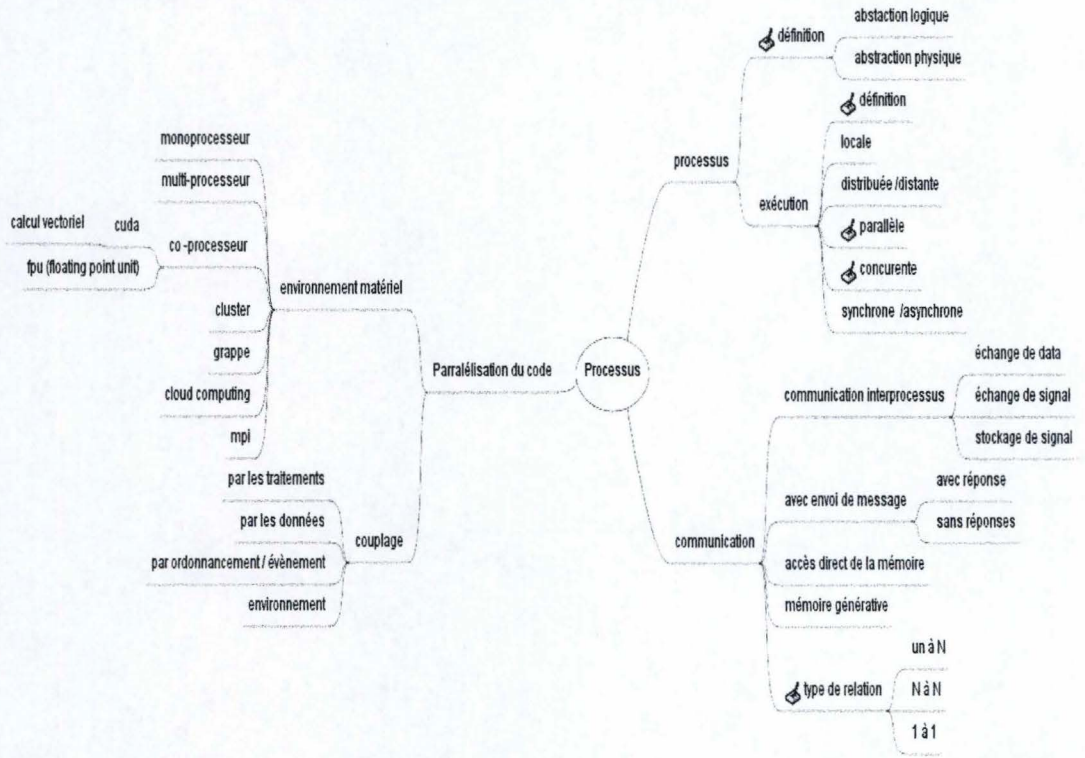
Les concepts

Chapitre 1

Linda

1.1 Introduction

Dans cette partie nous allons découvrir le modèle Linda. Linda est un modèle de calcul parallèle. Nous n'allons pas faire ici l'historique ni aborder tous les concepts de ce domaine. Cela nous conduirait en dehors des limites de notre mémoire. Comme l'illustre la carte mentale ci-dessous, nous pouvons constater que le domaine est très vaste et que les axes sont nombreux.



Toutefois, il nous semble important de nous arrêter quelques instants sur les notions de processus, de système parallèle, de système concurrent et de système distribué.

1.1.1 Les processus

Parmi la multitude de définitions que l'on peut trouver, nous en retiendrons deux données respectivement par Zaffalon et par Printz. Nous pouvons interpréter cette multitude comme la complexité à définir ce qu'est un processus en informatique.

Printz définit le processus dans un contexte plus général comme étant "une unité élémentaire, séquentielle par définition, dont la configuration est connue". [Printz, 2009]

Quant à Zaffalon, il déclare que : "l'association d'un objet possédant un comportement autonome et d'un algorithme décrivant son comportement sera appelée processus séquentiel ou thread". [Zaffalon, 2007, p. 8] Cette définition est donnée dans un contexte objet et plus particulièrement dans un livre consacré à la programmation concurrente en temps réel.

Dans les deux définitions, nous retrouvons l'idée de séquence qui, pour Printz, est inhérente à ce qu'il appelle "une unité élémentaire".

Zaffalon, quant à lui, introduit une notion plus moderne à notre sens qui est celle d'autonomie. En effet, nous pensons que la notion de séquence est intimement liée, d'abord à l'abstraction du langage que l'on utilise, ensuite à l'implémentation hardware et, en dernier lieu, à l'algorithme employé. En outre, il est souvent difficile de savoir comment va être exécuté ce que nous croyons être une séquence. Pour illustrer notre conviction, il suffit de regarder une multiplication matricielle. Celle-ci peut en effet se traduire soit par un processus séquentiel soit par un processus parallèle, distribué ou même concurrent.

En conclusion, nous insisterons davantage sur la notion de processus autonome plutôt que sur celle de processus séquentiel.

Notons que le terme de **processus communiquant** était également intéressant mais il est malheureusement historiquement lié au modèle CSP (Communicating Sequentials Processes). [Hoare, 1978] Les CSP sont un formalisme pour la conception et l'analyse des systèmes parallèles.

1.1.2 Systèmes parallèles

D'après Zaffalon, un système parallèle est un système qui exécute soit conceptuellement soit physiquement deux processus strictement séquentiels. [Zaffalon, 2007, p. 9]

1.1.3 Systèmes concurrents

Toujours d'après Zaffalon, un système concurrent est un système dont les processus, qui s'exécutent de manière parallèle, partagent les mêmes ressources. [Zaffalon, 2007, p. 9] "Le terme de concurrent, qualifiant un processus ou thread, tire son origine du fait que les threads sont en compétition, en concurrence entre eux pour l'obtention de la ressource processeur. [Zaffalon, 2007, p. 11]

1.1.4 Systèmes distribués

On parlera ici plus particulièrement d'architecture distribuée. D'après Wikipedia [Wikipedia, 2009b], cette dernière notion fait davantage appel au type de communication qu'utilisent les systèmes autonomes pour accomplir une tâche.

Après cette très courte introduction sur les processus et leur environnement d'exécution, nous allons présenter la naissance de Linda, le langage de coordination que nous avons choisi d'implémenter.

1.2 Historique de Linda

En 1989, Nicholas Carriero et David Gelernter, deux chercheurs travaillant à l'Université de Yale aux Etats-Unis, publient *Linda in context* dans Communications of ACM [Carriero et Gelernter, 1989]. Cet article est fondateur du langage de coordination Linda.

Dans cet article, les auteurs expliquent avec un certain triomphalisme la supériorité de leur modèle par rapport aux modèles de l'époque que sont le message passing, la programmation objet concurrente, les langages logiques concurrents et la programmation fonctionnelle. "Linda est un modèle plus simple, plus puissant et plus élégant que ces quatre alternatives" [Carriero et Gelernter, 1989, p. 444] annoncent-ils d'emblée.

Ils veulent avant tout convaincre le lecteur que l'on ne peut écrire des programmes parallèles sans jeter un coup d'oeil du côté de Linda. Cet article semble être une réponse aux *critiques* que certains formulent à l'égard du modèle Linda.

Pour parvenir à leurs fins, les auteurs comparent leur modèle avec celui de leurs quatre concurrents. Certes ils présentent leur modèle mais c'est surtout une comparaison pratique. On devine la syntaxe de Linda mais la sémantique des différents concepts est absente.

Quoi qu'il en soit, cet article permet de dégager les concepts de base de Linda. En outre, la polysémie des concepts, bien qu'elle soit gênante du point de vue mathématique, peut être intéressante. En effet, nous pouvons considérer ce modèle mal défini comme une heuristique qui donnera naissance à de nouveaux modèles et qui incitera les chercheurs scientifiques à plus de rigueur dans la création des nouveaux modèles. Ils tenteront ainsi d'apporter une meilleure sémantique aux primitives de base.

Bien que nous puissions trouver des exemples plus concrets relatifs à l'utilisation de Linda et plus particulièrement de C-Linda dans [Carriero et Gelernter, 1990], nous examinerons les concepts de base que cet article explique.

1.3 Contexte historique de cet article

A l'époque de la publication de cet article, l'informatique était surtout utilisée pour la gestion ou pour le calcul. Nous n'étions pas encore dans la période où l'informatique est la science de l'information.

Pour la parallélisation des calculs et donc la communication entre les processus, les informaticiens de l'époque faisaient appel à deux paradigmes qui sont :

- le partage de données dans lequel nous trouvons par exemple les sémaphores et les moniteurs ;[1]
- l'envoi de messages dans lequel se situent notamment les RPC (Remote Procedure Call).

Nous devons les sémaphores à E. Dijkstra et les moniteurs à Hoare.

Toutefois, ces deux paradigmes ne permettent pas de rendre compte de tous les problèmes de coordination. Ainsi en est-il de la problématique des agents mobiles ou, plus simplement, de celle des technologies Web.

[1] Le lecteur intéressé pourra trouver un historique ainsi que l'implémentation des moniteurs en Java dans [Zaffalon, 2007, chapitre 8].

1.4 Les concepts historiques

Les concepts repris ci-avant apparaissent aussi de manière plus explicite dans l'ouvrage "How to write parallel programs" rédigé par les mêmes auteurs. [Carriero et Gelernter, 1990]

"Linda est constitué de quelques opérations de base qui incarnent le modèle de programmation parallèle qu'est l'espace de tuples."

Ces opérations de base sont :

- out : génère un tuple data dans l'espace de tuples ;
- eval : génère un tuple dans l'espace de tuples ;
- in : consomme un tuple dans l'espace de tuples ;
- rd : lit un tuple dans l'espace de tuples.

On peut regrouper en deux catégories les opérations de base : celles qui génèrent (créent) un tuple (out, eval) et celles qui extraient l'information de l'espace de tuples (in, rd).

Ces opérations de base agissent sur un **espace de tuples**. L'espace de tuples agit comme un médiateur entre tous les processus.

On peut supposer que toutes les opérations du modèle de base sont ACID :

- Atomique : la transaction est entièrement exécutée ou pas du tout ;
- Consistance : l'état du système est laissé cohérent après l'exécution d'une transaction ;
- Isolation : une transaction ne voit que ses états intermédiaires et non les états intermédiaires des autres transactions ;
- Durable : toute modification des données doit être explicite.

"Linda est un modèle de création de processus et de coordination qui est orthogonal au langage de base dans lequel il est incorporé."

Linda ne s'occupe pas du calcul proprement dit : il laisse cette tâche au langage de calcul qu'il parallélise en lui adjoignant ces quelques opérations de base sur un espace de tuples.

"Ce modèle est basé sur la communication générative."

Il explique que si deux processus veulent communiquer, ils n'ont pas besoin de partager un message ou une variable : il leur suffit de générer une donnée (un tuple) et de la mettre dans un espace de tuples (TupleSpace). Une fois dans l'espace de tuples, cette donnée est indépendante du ou des processus qui l'ont générée et elle est accessible par tous les processus sans distinction.

Clairement ici les deux processus ignorent tout l'un de l'autre. De plus, on perçoit que la communication est bien asynchrone : il n'est pas nécessaire de coordonner le moment où l'information sera disponible mais bien de l'utiliser quand elle sera disponible.

"...un processus qui a besoin de créer un deuxième processus qui s'exécute de manière concurrente génère un "tuple vivant"."

La création de nouveaux processus est elle aussi assurée par le même mécanisme que pour les données. Une primitive indique simplement la nécessité de créer un ou plusieurs processus pour évaluer le tuple. Cette primitive est eval.

"Nous pouvons créer une collection de tuples .... qui seront accessibles à plusieurs processus simultanément."

Notons que si nous comprenons bien que les informations sont accessibles, nous ne voyons pas

clairement si elles sont atomiques, ni quelles sont les primitives concernées par cette possibilité, ni comment est gérée la file d'attente (fifo ou lifo).

Que se passe-t-il en cas de deadlock ? Comment peut-on les détecter ? Qui les gère : est-ce l'espace de tuples lui-même ou un programme de monitoring externe ?

Bref, en cas d'implémentation, on peut deviner la difficulté mais aussi les choix personnels qui sont possibles. On pourrait croire que les articles parus postérieurement vont éclaircir la situation sur la manière de les implémenter. Si certains vont effectivement proposer des solutions, notons que le manque de cohérence sémantique introduira des situations difficilement gérables.

En résumé, nous pouvons conclure que Linda est un concept qui permet la cohabitation *"pacifique"* des programmes qui l'accueillent et qui se focalise uniquement sur la communication, la création de processus et la coordination. Les auteurs d'ailleurs reviendront de manière plus explicite sur cette séparation entre le calcul et la coordination dans l'article *Coordination Languages and their significance* [Gelernter et Carriero, 1992]. Ils fondent leur point de vue sur leur expérience de terrain et expliquent pourquoi il faut séparer la coordination du calcul.

1.5 Les concepts

1.5.1 Les tuples

Un tuple est une suite ordonnée de champs. Chaque champ est typé. Ce typage est évident pour les tuples datas. Un tuple data est un tuple qui ne contient que le contenu de variables qui sont évaluées. Quant aux tuples vivants qui sont donc des processus qui seront évalués, c'est le type que l'on obtient en retour.

Un tuple data ou tuple [2] s'écrit de la manière suivante : ("une chaîne", 13,14.56). Ce tuple contient trois champs qui sont un champ chaîne (string), un champs entier (int) et un champs réel (float).

Voici un exemple d'émission en 'C' :

```
– out ("%s*%d", "Ceci est un test", j);
```

Et le même en Python :

```
– Ts_out ("Ceci est un test", j)
```

Un tuple vivant s'écrit de la même manière : ("une chaîne", sin(90)). Son évaluation produira le tuple data suivant : ("une chaîne", 1).

1.5.2 Les patrons

Comme un tuple, un patron est une suite ordonnée de champs. Chaque champ contient soit une valeur typée, soit un type, soit une expression régulière. La notation des patrons[3] correspond bien souvent aux possibilités du langage d'accueil et aux choix faits par l'implémenteur.

Voici quelques notations que l'on rencontre :

– (string,12,none) ;

[2]Par convention un tuple sans précision est toujours un tuple data.
[3]Templates en anglais.

- (string,int,*) ;
- (?string, ?int,12) ;
- (?o ?o,12,*).

Comme on peut le voir, les variations de syntaxe sont inhérentes au langage d'accueil.

La relation qui existe entre le tuple et le patron est l'opération d'unification : ("bonjour", 12, 12) s'unifie avec (string, 12, none) si la signification que l'on donne à none est bien n'importe quel type.

La condition nécessaire pour l'unification est que le nombre d'éléments du tuple et du patron soit identique.

La condition suffisante pour l'unification est que, pour tout élément $n_i \in$ Ts et pour tout élément p du patron, nous ayons la relation :

$type(n_i) = type(p_i) \vee n_i = p_i$

Notons que si un patron peut s'unifier avec plusieurs tuples, un seul tuple est choisi arbitrairement.

Ce mécanisme d'unification est propre au mécanisme de mémoire **associative** qui caractérise le modèle Linda. En effet, les processus ont accès au contenu des tuples par **association** ou par **unification**.

1.5.3 L'espace de tuples

L'espace de tuples est un concept hérité du concept de tableau (dashboard). C'est une mémoire virtuelle partagée. L'espace de tuples est généralement dénoté Ts. Il accueille les tuples. Selon les implémentations, nous pouvons avoir un ou plusieurs espaces de tuples. L'espace de tuples est géré par les primitives du langage de coordination. Aux primitives de base se sont ajoutées, selon les implémentations, de nouvelles primitives.

Voici une brève description de l'ensemble des primitives couramment utilisées.

1.5.4 Les primitives de bases

Les primitives de base suivantes donnent accès à l'espace de tuples :

- out(tuple) permet d'ajouter un tuple à l'espace de tuples ;
- eval(tuple) ajoute un tuple dans l'espace de tuples et provoque en cas lecture son évaluation ;
- in (tuple) lit et consomme un tuple de l'espace de tuples. Cette primitive bloque l'opération tant qu'elle n'a pas consommé le tuple ;
- rd (tuple) lit un tuple (en vérifiant qu'il existe) de l'espace de tuples. Cette primitive bloque l'opération tant qu'elle n'a pas effectivement lu le tuple ;
- tsc(TupleSpace) crée un nouvel espace de tuples.

1.5.5 Les primitives supplémentaires

Ces primitives sont pour certaines apparues, comme nous le verrons par la suite, pour clarifier ou pour combler les problèmes de sémantique du modèle de base. D'autres ont été ajoutées pour compléter le modèle (Machine de Turing) :

- inp(tuple) vérifie la présence du tuple dans l'espace de tuples et, dans l'affirmative, le consomme ;

- rdp(tuple) vérifie la présence du tuple et, dans l'affirmative, lit la valeur ;
- copy(TupleSpace, tuple) copie les tuples d'un espace dans un autre espace de tuples en effaçant les tuples d'origine ;
- copy-collect(TupleSpace, tuple) copie les tuples d'un espace dans un autre espace de tuples en préservant les tuples d'origine.

1.5.6 La syntaxe

Selon le language, Linda adoptera la syntaxe du language hôte.

1.5.7 Grammaire et sémantique

Comme nous l'avons vu, le modèle de départ avait été peu défini. Cela a donné lieu à de multiples interprétations et à de multiples tentatives pour exprimer une univocité des concepts.

Pour décrire et définir avec précision le modèle Linda, il fallait des outils conceptuels. Pour définir les opérateurs de base, les sémantiques dénotationelles, axiomatiques, algébriques et opérationelles ont été utilisées. Notons toutefois que c'est souvent la sémantique opérationelle qui a été choisie, ce qui peut paraître contradictoire lorsque l'on connait le manque de consensus sur ce que faisait réellement telle ou telle primitive.

Nadia Busi et al. dans [Omicini *et al.*, 2001] définissent la grammaire et la sémantique des agents (ou processus) en utilisant les multisets et une variation de la "Chemical Asbstract Machine" (CHAM) définie par Gérard Berry et Gérard Boudol[Berry et Boudol, 1992]. CHAM est un modèle qui peut définir un système concurrent ou parallèle en se basant sur la métaphore des réactions chimiques. Les auteurs utilisent les termes de solution, de molécule et de réaction pour décrire les changements d'état des systèmes distribués. La sémantique est de type opérationnelle.

1.5.8 Grammaire

Nadia Busi et al. définissent la grammaire pour les quatre primitives de base. Ils utilisent un set d'agents et un set de messages qu'ils appellent Datas est qui sont constitués de $a, b, \ldots$:

$$A ::= 0 | \mu.A | \sum_{i \in I} A_i | K$$

$$\mu ::= out(a) | in(a) | rd(a) | eval(A)$$

TAB. 1.1 – Grammaire de Linda

avec :
- μ dénote une instance parmi les primitives de coordination ;
- K est le nom générique pour un agent du set des noms d'agent avec comme relation K=A ;
- 0 dénote un agent qui ne peut rien faire ;
- $\mu.A$ est un agent qui exécute la primitive μ et qui ensuite se comporte comme un agent A ;
- $\sum_{i \in I} A_i$ est un agent dont la somme des actions est la composition de chaque action.

La sémantique

Toujours d'après Nadia Busi et al., " un système est composé d'un mutiliset d'agents actifs et d'un multiset de messages disponibles".

"Formellement un système est une paire $(Ag, DS) \in \mathcal{M}(Agents) \times \mathcal{M}$(Données) (où $\mathcal{M}(Set)$ est utilisé pour dénoter le set des multisets des éléments qui composent le Set)."[Omicini *et al.*, 2001, page 10] Où $\mathcal{M}(Agents)$ sont définis par A, A', ...

Dans le tableau suivant qui reprend la sémantique proposée par [Omicini *et al.*, 2001, page 11], le symbole $\oplus$ est utilisé pour dénoter l'union d'un multiset. Dans le cas d'un singleton, les { } sont omises.

$$(out(a).A \oplus Ag, DS) \rightarrow (A \oplus Ag, DS \oplus a)$$

$$(in(a).A \oplus Ag, DS \oplus a) \rightarrow (A \oplus Ag, DS)$$

$$(rd(a).A \oplus Ag, DS \oplus a) \rightarrow (A \oplus Ag, DS \oplus a)$$

$$(eval(A').A \oplus Ag, DS) \rightarrow (A' \oplus A \oplus Ag, DS)$$

$$\frac{(Aj, DS) \rightarrow (A', DS')}{(\sum_{i \in I} A_i \oplus Ag, DS) \rightarrow (A' \oplus Ag, DS')} \qquad sij \in I$$

$$\frac{(A, DS) \rightarrow (A', DS')}{(K \oplus Ag, DS) \rightarrow (A' \oplus Ag, DS')} \qquad siK = A$$

TAB. 1.2 – Sémantique de Linda

Après avoir présenté ce tableau, Busi et al. livrent quatre remarques que nous exposons succintement :

- ils n'ont pas modélisé l'unification des tuples et des patrons mais, disent-ils, cela ne pose aucun problème ;
- la primitive rd(a) pourrait être sémantiquement équivalente à in(a) suivi immédiatement d'un out(a) si nous faisions l'hypothèse de la simultanéité des opérations ;
- la primitive out(a) peut posséder différentes sémantiques selon que l'on se place du point de vue de l'instantanéité ou de la sérialisation (ordonnée ou non) de l'exécution des opérations ;
- on peut prouver que ces quatre opérations ne permettent pas d'avoir une machine de Turing complète.

1.5.9 La création d'espaces de tuples

Le modèle initial ne possède qu'un seul tuple : la primitive pour la création des espaces de tuples est supposée exister dans [Butcher *et al.*, 1994].

1.5.10 La sémantique du copy-collect

Dans [Rowstron et Wood, 1996], les auteurs vont présenter le problème que pose la sémantique de rd. En effet, il est impossible de s'assurer lors de l'utilisation de cette requête que l'on va parcourir l'ensemble des informations qui correspondent à la requête.

Rd ne peut déterminer que la présence d'une information. Si cette information est un singleton, il n'y pas de problème. Par contre, dans tous les autres cas, il est impossible de parcourir une liste d'objets tout en s'assurant que l'on atteint chaque élément de l'ensemble.

Pour illustrer cette problématique, Rownstron et Wood prennent comme exemple une composition de relation binaire.

Ils proposent donc une primitive copy-collect [Rowstron et Wood, 1996] qui copie les tuples d'un espace de tuples dans un autre espace de tuples. Elle est donc non destructive.

1.5.11 Une nouvelle sémantique pour inp

Jacob et Wood proposent dans [Jacob et Wood, 2000] une nouvelle sémantique pour inp.

Il s'agit d'une opération qui peut dans certains cas rendre diffus les deadlocks. En effet, le système étant asynchrone, il est plus simple de surveiller les deadlocks que de mettre à disposition une primitive qui pourrait les dissimuler. Voici par exemple le cas classique du deadlok en Linda avec :

Agent X	Agent Y

`while True :`
`  u:=in(a)`
`  t:=inp(b)`
`  if not t:`
`  out(a)`
`  else:`
`  job()`
`  break`
	`while True :`
	`  in(b)`
	`  t:=inp(a)`
	`  if not t:`
	`  out(b)`
	`  else:`
	`  job()`
	`  break`

Les auteurs constatent l'utilisation de la primitive inp dans deux cas :

- le premier pour éviter d'être bloqué dans l'exécution du programme ;
- le deuxième pour détecter la fin d'un processus.

Selon eux, la première situation n'est pas du ressort des implémenteurs de Linda. Il suffit en effet de créer des threads pour éviter qu'une opération ne bloque le déroulement d'un programme.

Ils estiment par contre que la seconde situation est plus intéressante. Ils considèrent que cette situation peut être utilisée comme une technique de *point fixe*. Pour ce faire, ils proposent d'utiliser la primitive inp, qui n'est pas bloquante à l'origine, comme une primitive bloquante. Cette manière de faire va provoquer des deadlocks. A partir de ce moment-là, il suffit au système de détecter les deadlocks et donc de signaler la fin du processus.

Les deux auteurs expriment toutefois le fait que cette solution est coûteuse au niveau de l'implémentation.

Cette sémantique est implémentée dans une version de Linda, Pylinda, que nous avons testée.

Pour notre part, nous proposons aussi une solution qui nous semble moins coûteuse au niveau de l'implémentation et que nous exposerons dans la troisième partie du mémoire.

1.5.12 La sémantique du collect

Contrairement à la primitive copy-collect introduite par Rowstron et
Wood [Rowstron et Wood, 1996], cette primitive déplace les tuples en les consommant. Elle a été introduite par [Butcher *et al.*, 1994].

Butcher et al. partent du principe que la création dynamique d'espaces de tuples est possible. Ils expliquent par un exemple la difficulté d'apporter une solution à un problème particulier qui est la réalisation d'une sommation en parallèle maximisée. La première solution consiste à exécuter deux in pour les opérandes et un out pour la sommation. On suppose que les données arrivent de manière séquentielle dans l'espace de tuples.

Butcher et al. passent alors en revue les possibilités qu'offrent les primitives existantes. Ils proposent même de l'implémenter en utilisant la primitive inp pour l'espace de tuples d'origine et out pour l'espace de tuples de destination.[4]

Butcher et al. de conclure alors qu'aucune primitive ne peut convenir et qu'il est impératif de créer une nouvelle primitive "collect".

Cette démarche est souvent reprise dans les articles que l'on trouve sur Linda :

- on énonce un problème particulier ;
- on apporte une solution particulière.

Notons qu'il est rarement fait référence aux autres primitives et à l'impact d'une nouvelle primitive. Les primitives sont toujours évaluées ou présentées de manière isolée et non réellement contextualisée par rapport au modèle ou au langage de coordination qu'est Linda. La vision globale et cohérente du modèle ne semble pas être une préoccupation majeure des chercheurs de l'époque.

1.5.13 Accès exclusif à l'espace de tuples

Notre implémentation permet à un programme de créer une région critique en ayant un accès exclusif aux espaces de tuples. Cela peut être nécessaire si l'on souhaite s'assurer qu'une précondition existe sur un ou plusieurs espaces de tuples. Cela peut aussi servir pour éviter certains interblocages. Nous n'avons pas trouvé au cours de notre lecture la moindre trace d'un mécanisme semblable. Mais il nous apparaissait pourtant utile voire nécessaire dans certaines situations.

1.5.14 Le dîner des philosophes

Pour illustrer les primitives du langage Linda, nous allons présenter un exemple que nous avons trouvé chez [Carriero et Gelernter, 1990] et qui est repris sous le tableau 1.3 page 24. Cet exemple est le problème des philosophes énoncé avec un humour contenu par Dijkstra. En voici la traduction.

"Cinq philosophes numérotés de 0 à 4 habitent une maison où une table est dressée pour eux. Chaque philosophe possède sa place à table. Leur seul problème, en plus de ceux de la philosophie,

[4]Remarquons que la définition de ce qu'est une transaction nous semble ici très important. En effet, pour une totale équivalence sémantique, il faut que lors de l'inp l'accès à l'espace de tuples d'origine soit exclusif. En effet, la sémantique de collect est qu'il copie tous les tuples présents du tuple d'origine lors de la requête.

```
phil(i)
   int i;
   {while(1)
      {
         think();
         in("room ticket");
         in("chopstick",i);
         in("chopstick", (i+1)%Num);
         eat();
         out("chopstick",i);
         out("chopstick", (i+1)%Num);
         out("room ticket");
       }
   }


{
   int i;
   for(i = 0;i<Num;i++)
      {
      out("choptick",i);
      eval(phil(i));
      if(i <(Num-1)) out("room ticket");
      }
}
```

TAB. 1.3 – Le diner des philosophes

c'est que le plat qui est servi est une variété très spéciale de spaghettis qui doit être mangée avec deux fourchettes. Il y a une fourchette entre chaque assiette, ce qui ne présente pas de difficulté mais qui a pour conséquence que deux voisins ne peuvent pas manger simultanément." [Dijkstra, 1971].

1.5.15 Le modèle master-worker

Le modèle Linda est souvent utilisé avec le modèle master-worker. Un processus principal (le master) va ordonnancer et contrôler l'exécution des tâches effectuées par les workers. Il peut y avoir autant de workers que l'on souhaite. Cette manière de faire est assez pratique pour l'élaboration de programmes distribués ou parallèles.

Elle est si fortement ancrée que les programmes sont constitués d'un seul listing qui renferme à la fois le code pour le master et le code pour le worker. C'est par passage de paramètres lors de l'exécution du programme que l'on va décider quelle partie de code va s'exécuter.

Si dans notre mémoire nous nous focaliserons principalement sur le modèle master-worker, signalons que Linda a aussi donné naissance à d'autres modèles que nous envisagerons de manière théorique. Citons par exemple le modèle agent où chaque agent est autonome et communique via un ou des TupleSpaces qui peuvent être distribués.

On le voit, le modèle Linda est assez riche et permet une architecture logicielle assez diversifiée.

1.5.16 Division du travail

Armé du modèle master-worker, la division du travail reste encore à définir. Le modèle de coordination Linda est souple. Voici, sans être exhaustif et en nous basant sur le domaine du traitement d'images, les possibilités offertes en prenant le point de vue fichier à traiter :

- chaque pixel d'une seule image est traité par un worker ;
- l'image est découpée en plusieurs parties et chaque worker exécute un traitement constitué de plusieurs pixels ;
- une image est traitée dans sa globalité par un worker.

Nous pouvons aussi prendre le point de vue du programme :

- le worker exécute une fonction de traitement ;
- le worker exécute un sous-ensemble du programme ;
- le worker exécute un programme.

Comme nous pouvons le remarquer les choix et les avantages sont nombreux. L'utilisation de Linda permet d'équilibrer naturellement la charge du processeur. En effet, chaque worker exécute le travail selon ses possibilités sans se soucier le moins du monde des autres. Il n'est pas nécessaire pour le programmeur de déterminer la puissance de calcul que l'on va attribuer à tel ou tel traitement en fonction de la puissance du processeur. On le voit, le modèle Linda peut être implémenté dans une architecture distribuée où les puissances de chaque composant peuvent être différentes et non équilibrées.

Toutefois, il faut être attentif à la manière de procéder. Il faut d'abord bien distinguer la partie du processus qui peut être parallélisable de celle qui ne peut pas l'être. Après cette démarche, il est important de mettre au point des traitements qui minimisent :

- les goulots d'étranglement dus aux transferts d'informations ;
- les temps d'attente de synchronisation des processus via l'espace de tuples ;
- les responsabilités des processus workers ;
- les tâches de contrôle du master.

Dans la partie labo, nous expérimenterons les différentes manières de procéder et nous essayerons de dégager quelques pistes pour élaborer des stratégies gagnantes.

1.5.17 Les implémentations

Les implémentations se trouvent principalement dans deux catégories :
- celles qui implémentent le modèle initial ;
- celles qui étendent le modèle initial.

Les implémentations Linda sont nombreuses. Chaque langage a vu naître une implémentation : C, Fortran, C++, Lisp, Java, Python, Ruby, pour ne citer que quelques-uns.

C-Linda ou Fortan-Linda sont les implémentations d'origine.

Parmi les plus connues du modèle intial et qui sont supportées par des sociétés commerciales, on trouve :
- TCP-Linda de Scientific Computing Associates [5] ;
- JavaSpaces développée par SUN [6] et incorporée au sein de leur projet JINI ;
- TSpaces développée par IBM [7].

Le modèle Linda a également donné lieu à plusieurs implémentations universitaires de part le monde :

- Tucson ;
- Klaim ;

[5] www.lindaspace.com
[6] http ://www.jini.org/wiki/JavaSpaces_Specification
[7] http ://www.almaden.ibm.com/cs/TSpaces.

- Mars.

1.5.18 Modèle et langage de coordination

Il est important de distinguer un **modèle de coordination** d'un **langage de coordination**. Mais nous avons constaté au fil de nos lectures que l'un et l'autre étaient employés indifféremment par abus de langage.

Un modèle de coordination

Dans [Gelernter et Carriero, 1992, p. 92], "un modèle de coordination est la colle qui assemble des activités séparées en ensemble". Un modèle de coordination peut être vu comme un triplet (E,L,M) où E représente les parties qui doivent être coordonnées, L le média qui est utilisé pour coordonner les entités et M la sémantique du framework auquel il adhère [Papadopoulos et Arbab, 1998, p. 4]. Nous retrouvons d'ailleurs cette même formulation du modèle dans [Omicini *et al.*, 2001, p 7 et 8] même si elle est formulée en terme d'agents (ou de processus).

Un langage de coordination

Dans [Gelernter et Carriero, 1992, p. 92], "un langage de coordination incarne un modèle de coordination et procure les outils pour créer les processus et gérer les communications entre ces processus".

1.5.19 Taxonomie des langages de coordination

1.5.20 Vers une classification

Le modèle Linda a donné naissance à une multitude d'implémentations. Les primitives du modèle de base ont été redéfinies mais le modèle aussi, donnant lieu à de nouvelles perspectives de coordination.

Il est tentant de vouloir catégoriser ces différentes implémentations. Beaucoup d'auteurs ont essayé de définir ainsi des critères comme par exemple le fait qu'un langage implémentait de manière implicite (endogène) ou explicite (exogène) les directives de coordination. Notons la difficulté d'une telle démarche puisque beaucoup d'implémentations proviennent d'équipes universitaires qui redéfinissent chacune le domaine et apportent des solutions à leur modèle conceptuel. Cette diversité permet donc de choisir autant de critères pertinents : le paradigme du langage d'accueil (impératif ou non impératif) ou le domaine auquel il s'adresse (l'intelligence artificielle).

Nous sommes en fait très loin d'un standard industriel comme pourrait l'être la définition du C ou du Fortran.

Le projet Accord [Accord, 2002] propose une piste de taxonomie basée sur les solutions qu'apporte un modèle de coordination, à savoir :

- la gestion concurrente des ressources ;
- la gestion de l'information ;
- la synchronisation entre les différentes activités ;
- les décisions de groupes.

Nous constatons que, selon le projet Accord, la coordination va bien au-delà d'un simple modèle client-serveur ou d'une simple configuration master-worker.

Le projet Accord propose ainsi une approche orientée composants. Ces derniers encapsulent les primitives de coordination. On trouve dans cette catégorie Flo/c, CoLas et Oblog.

Dans ce rapport Accord, nous trouvons aussi une taxomie proposée
par [Papadopoulos et Arbab, 1998].

Dans [Papadopoulos et Arbab, 1998], les deux auteurs classifient les modèles de coordination en deux classes : les langages orientés données et les langages orientés événements (orientés processus ou orientés tâches). La grande différence entre les deux modèles réside dans la séparation totale entre la coordination et le calcul pour les modèles orientés datas [Papadopoulos et Arbab, 1998, p. 5].

Dans les modèles **orientés évènements**, les entités exposent des interfaces. Les entités coordonnées sont dans une relation producteur-consommateur. Les interfaces sont composées de connecteurs de type entrée (input) ou de type sortie (output) qui permettent la communication point-à-point entre deux entités. Le broadcast entre une et plusieurs entités est possible.

Dans le premier groupe on trouve Linda, BauHaus Linda, Laura et Sonia, et dans le second PCL (Proteus Configuration Language), Darwin/Regis ou Durra, et MANIFOLD.

Comme le modèle orienté datas est expliqué en long et en large dans notre mémoire, nous ne décrirons qu'un modèle de coordination orienté événements.

Un modèle orienté événements

Pour exemple dans [Schumacher *et al.*, 1998], l'article introduit quelques concepts du modèle événementiel STL. Celui-ci est constitué d'objets soit agglomérés en blobs soit typés. Les blobs comme les objets exposent des ports qui sont les interfaces pour communiquer avec d'autres blobs ou processus. Les sémantiques des connexions sont :

- point-à-point :
  - 1 à 1 ;
  - 1 à n ;
  - n à 1 ;
- groupe clos : diffusion d'un message à tous les membres d'un groupe ;
- tableau noir : les messages sont placés sur un tableau noir pour être utilisés par plusieurs processus. Au lieu de partager des données ce sont bien des événements qui sont mis en commun.

Ajoutons également les évènements selon [Schumacher *et al.*, 1998] :

- accessed(p) : le port a été accédé ;
- unbound(p) : pas de partenaire de communication ;
- isempty(p) : ne contient pas de donnée ;
- isfull(p) : le port est plein ;
- msg_handled(p, int n) : n messages gérés ;
- less_msg_handled(p, int n) : $\leq$ n messages gérés.

1.5.21 Les modèles à mobilité de code

Linda a aussi inspiré des équipes qui ont proposé des modèles où le code migre d'environnement en environnement pour être exécuté.

1.5.22 Les critiques du modèle

Le modèle est de prime abord attrayant. Il a changé notre vision des manipulations de données dans des contextes distribués.[Ericsson-Zenith, 1992] Mais après plus de vingt ans, tient-il toujours la route ?

En une vingtaine d'années, tous les domaines de l'informatique ont considérablement évolué :
- les architectures multiprocesseurs se sont vulgarisées ;
- les compilateurs sont devenus de plus en plus performants ;
- les systèmes d'exploitation ont eux aussi évolué ;
- les architectures logicielles se sont complexifiées ;
- les supercalculateurs intègrent aujourd'hui du routing de messages ;
- le calcul vectoriel et matriciel à l'aide des GPU est devenu accessible à Monsieur Tout Le Monde.

En un mot, si le modèle Linda avait déjà ses détracteurs, qu'en est-il aujourd'hui ? Nous n'allons pas répondre ici à cette question. En effet, elle sort du cadre de notre mémoire. Mais pour ouvrir le débat, nous allons reprendre quelques remarques formulées par Ericsson-Zenith [Ericsson-Zenith, 1992, pp 79-82] lors de sa défense de thèse de doctorat, soit trois ans après le premier article de Carreiro et Gelernter :

- il n'est pas possible a priori de connaître la nature d'un tuple par un processus ;
- l'optimisation lors de la compilation est difficile ;
- il n'y pas d'échange de variables par référence ;
- il est difficile de connaître a priori l'efficacité d'un programme Linda ;
- la performance d'un programme Linda n'est pas transposable à une autre configuration système.

1.5.23 Conclusions

Le modèle Linda est une modèle qui est riche par la simplicité de ses concepts.

Il a donné naissance à de nombreux projets tant au niveau du calcul parallèle que de la mobilité du code ou de la communication entre les processus ou agents.

Si le modèle tentait déjà il y a vingt ans de trouver une solution élégante, puissante et simple à la fois au domaine complexe et ardu du calcul parallèle et/ou distribué, le challenge est loin d'être démodé aujourd'hui.

La puissance des processeurs s'exprime actuellement davantage par le nombre de coeurs qu'ils peuvent gérer et par le nombre d'opérations vectorielles qu'ils peuvent effectuer. La communication interprocessus est, comme on peut le voir, encore plus d'actualité qu'il y a vingt ans.

Mais la qualité d'un modèle se mesure aujourd'hui par sa cohérence et par sa possible standardisation. Or il apparaît souvent difficile de standardiser sans une base saine et cohérente des concepts.

Si on ne peut prouver les éléments d'un modèle, il faut au moins se mettre d'accord sur sa sémantique.

Nous terminons cette partie des concepts de notre mémoire en abordant le langage de programmation que nous avons décidé d'associer à Linda.

Chapitre 2

Python

2.1 Introduction

Dans ce chapitre, nous allons expliquer pourquoi nous avons choisi Python comme langage de calcul pour explorer Linda. Nous passerons en revue les choix possibles puis nous explorerons les possibilités offertes par le langage Python.

2.2 Choix d'un langage de calcul

Le choix d'un langage de programmation est délicat. La multitude des paradigmes et des implémentations ne manque pas. D'emblée, le choix d'un langage de scripting peut semer le doute. Est-ce vraiment un choix approprié dans le domaine du traitement du signal où l'on verrait plutôt le C, le C++ ou le Java ?

Pour mieux comprendre le choix d'un langage, nous devons conserver à l'esprit les objectifs du mémoire : fournir une boîte à outils en proposant, d'une part, des prototypes de traitement et, d'autre part, une implémentation Linda adaptée au domaine du traitement.

La tentation est grande de choisir pour chaque domaine le language le plus approprié car, en fait, nous avions deux axes fort différents :

- expérimenter des algorithmes de traitement du signal ;
- implémenter une solution client-serveur.

Pour le traitement du signal, une boîte à outils tel que Matlab était très attirante. La documentation est abondante et les exemples ne manquent pas sur Internet. Quant au serveur, le C ou le C++ semble le plus approprié.

Mais notre objectif n'était pas de fournir les meilleurs outils en terme de performance mais bien de montrer les possibilités d'un langage de coordination appliqué à un domaine particulier. Nous privilégions en effet davantage la rapidité de développement et la flexibilité de l'implémentation plutôt que la vitesse pure. D'autre part, la gestion du projet nous semblait risquée si nous devions employer simultanément deux langages de développement.

Ajoutons à cela que nous avions des préférences au niveau des langages : nous souhaitions travailler avec un langage orienté objets qui offre une certaine souplesse et qui dispose d'outils dans le domaine que nous comptions explorer.

Java et C++ étaient de bons candidats pour la richesse de leurs bibliothèques dans le domaine d'étude mais ces deux langages ne sont pas les meilleurs candidats pour un prototypage rapide. Java est un langage très explicite et C++ demande un effort supplémentaire pour choisir les bibliothèques de base. Nous ne voulions pas passer du temps à choisir telle ou telle bibliothèque. Nous voulions aussi, si possible, échapper au cycle codage/compilation/exécution. Exit donc Java et C++.

Ajoutons à cela que nous voulions explorer des bibliothèques et visualiser rapidement les possibilités dans le domaine. Un langage de scripting nous paraissait intéressant. Ruby, Python ou Groove, proche de Java, nous tentaient. Une rapide enquête sur Internet terminait de nous convaincre de choisir Python pour les objectifs poursuivis dans ce mémoire. Il nous restait à convaincre que ce choix était judicieux ou, tout au moins, acceptable.

"Chaque langage a ses sujets de prédilection que ce soit pour des raisons historiques ou parce qu'il offre de réels avantages dans le domaine". [Ziadé, 2006, page 11]

Pour Python ces domaines de prédilection sont notamment le calcul scientifique et le prototypage rapide d'applications. Ayant déjà expérimenté Python régulièrement en tant qu'administrateur système, c'est un langage que nous apprécions car en plus de posséder des outils de base efficaces, il incite à une grande clarté du code. Nous connaissions les bibliothèques de base qui sont très riches et qui permettent, par exemple, de rapidement prototyper un serveur web ou un client ftp. Mais pour le calcul scientifique, c'était une véritable découverte.

Voyons maintenant quels sont les atouts du langage Python.

2.3 Python

Python est un langage multiparadigme : il peut être impératif, fonctionnel et objet.

C'est un langage interprété comme Java : il produit du byte code.

La gestion de la mémoire est automatique.

Le typage des données est dynamique. Python dispose de types de données évolués dont les nombres complexes, les sets, les xranges, les listes, les dictionnaires et les tuples. Il implémente notamment le tri sur les listes. Il gère nativement l'unicode même si cette gestion n'est pas parfaite.

Le code est portable puisque Python est porté sur la majorité des architectures actuelles.

Il en est aujourd'hui à la version 3.1. Cette version rompt la compatibilité avec les versions précédentes. À cause de cela, nous avons utilisé la version 2.5 qui est toujours largement utilisée. Signalons que la version 2.6 est une version de transit vers les versions 3.0.

Python dispose d'une bibliothèque standard qui lui permet d'offrir rapidement des programmes fonctionnels en peu de lignes : un serveur web, un client mail ou simplement la compression de fichiers ou l'accès à une base de données.

Python supporte les exceptions, le multi-threading, l'héritage multiple et la surcharge des opérateurs. Il est extensible, réflectif, introspectif, gratuit, libre, simple [Swinnen, 2009, in préface] ... et nous arrêterons-là l'énumération de ses qualités pour ne pas paraître partisan.

Bref, Python est un langage moderne qui permet de se concentrer sur l'algorithmique et sur sa traduction quasi immédiate sans se soucier d'une syntaxe compliquée ou sans chercher désespérément une bibliothèque qui fonctionne.

2.3.1 Les modes de Python

A l'instar des autres langages de script, Python offre deux modes : un mode interactif et un mode script. Le mode script est le mode où l'interpréteur exécute un script, tandis que le mode interactif est le mode qui permet à l'utilisateur d'introduire via une console son code ligne par ligne et de le voir évaluer immédiatement.

Pour une utilisation scientifique, nous pouvons employer la console **ipython**. Elle améliore les fonctionnalités de la console de base. Voici les fonctionnalités intéressantes :

- historique des commandes ;
- historique des résultats ;
- accès à l'aide en ligne de Python ;
- des commandes magiques ;
- la possibilité de définir des raccourcis (système d'alias) ;
- une complétion.

Ipython permet donc de tester et d'enregistrer très rapidement des séquences de code.

2.3.2 La documentation

Tant la communauté, la littérature spécialisée qu'Internet permettent d'avoir une information fiable et utilisable directement lorsque l'on code. De plus les tutoriels sont nombreux et de qualité.

Ajoutons à cela que Python propose la commande help(name) qui permet de manière interactive de connaître la documentation liée à la fonction, à la procédure ou à la classe que l'on manipule.

En d'autres mots, il est pratiquement impossible de se sentir seul quand on code en Python.

Une source importante de documentation est bien évidemment le site officiel de Python : `www.python.org`.

Nous vous recommandons la lecture des PEP qui permettent de comprendre en profondeur les choix posés ou les implémentations du langage Python.

2.3.3 Un dépôt logiciel

Il existe un dépôt central de packages pour Python. On peut simplement les installer par la commande **easy_install nomDuPackage**.

2.3.4 Les extensions scientifiques

Python offre un mécanisme assez simple pour wrapper une bibliothèque ou pour construire une bibliothèque dynamique. Cette facilité permet donc à de nombreux programmeurs de porter leur travail ou de l'interfacer avec Python. Pour conclure, Python dispose d'outils scientifiques de qualité. Citons par exemple :

- Scipy ;
- Numpy.

Quant au domaine que nous explorons, c'est-à-dire l'image et le son, nous trouvons un choix raisonnable d'outils ou de bibliothèques tels que :

- audio-lab ;
- pil ;
- vips.

Ces outils seront présentés dans la quatrième partie de notre mémoire.

2.3.5 Un langage de scripting pour la coordination

Dans [Omicini *et al.*, 2001], J.G Scneider et al. abordent la problématique de l'utilisation d'un langage de scripting pour coordonner des agents.

Cet article présentent deux applications du monde réel :

- CyberChair ;
- Picola Wiki.

Dans les deux applications c'est Python qui a été choisi.

Mais ne soyons pas non plus trop enthousiaste car, comme le font remarquer les auteurs, si les langages de scripting permettent de construire des applications évolutives et flexibles, ils doivent aussi apporter des améliorations quant à Dans [Omicini *et al.*, 2001, page 174] :

- leurs possibilités d'abstraction qui permettent difficilement d'implémenter des concepts de coordination de haut niveau ;
- leur architecture logicielle rigide ;
- leur gestion du modèle de concurrence qui est souvent gérée par une bibliothèque externe ;
- de meilleures possibilités pour implémenter plus de style de coordination que simplement des composants, des connecteurs et une définition des relations entre les composants ;
- une sémantique plus stricte qui permettrait une meilleure preuve des concepts.

Cette liste de souhaits est valable pour tous les langages de scripting en général. Pour Python, seul deux souhaits peuvent être retenus.

La composition modulaire de Python rend effectivement difficile dans certains cas la preuve formelle. Jusqu'il y a peu, les chaînes de caractères étaient gérées par une bibliothèque de la librairie de base.

La gestion de la concurrence est loin d'être aussi bien gérée qu'en Ada. Mais rappelons que si Linda n'est pas Ada, Python ne l'est pas non plus.

2.3.6 Le dîner des philosophes en Python

Ce code implémente en Python le dîner des philosophes. Il est conçu pour une architecture master-worker. Le worker initialise l'espace de tuples. On peut créer autant de workers que l'on souhaite.

Voici le code pour le master :

```
# -*- coding: utf-8

import linda as linda

if __name__ == '__main__':
```

```
    nPhilosophe = 10

    linda.connect()

    monPhilosophe = linda.universe._rd(("philosophe", linda.TupleSpace))[1]

    monPhilosophe=linda.TupleSpace()
    linda.universe._out(("philosophe", monPhilsophe))

    for couvert in xrange(0, nPhilosophe):
        monPhilosophe._out(("couvert", couvert))
        if couvert < (nPhilosophe-1):
            monPhilosophe._out(("ticket d'entrée"))
```

Voici le code pour le worker (le philosophe) :

```
# -*- coding: utf-8

#extension du path

import linda as linda

def penser():
    pass

def manger():
    pass

if __name__ == '__main__':

    linda.connect()
    monPhilosophe = linda.universe._rd(("philosophe", linda.TupleSpace))[1]
    monNom=monPhilosophe._in("nom",int)[1]
    nPhilosophe=monPhilosophe._rd(("nombrePhilosphe",int))

    while True:
        penser()

        monPhilisophe._in(("ticket d'entrée"))
        monPhilisophe._in(("couvert",(monNom) % nPhilosophe) # % est le modulo
        monPhilisophe._in(("couvert",(monNom+1) % nPhilosophe) # % est le modulo

        manger()

        monPhilisophe._out(("couvert",(monNom) % nPhilosophe) # % est le modulo
        monPhilisophe._out(("couvert",(monNom+1) % nPhilosophe) # % est le modulo
        monPhilisophe._out(("ticket d'entrée"))
```

Chapitre 3

La numérisation du signal

3.1 Introduction

Que ce soit dans le traitement de l'image ou dans le traitement du son, nous serons amenés à faire du traitement de signal.

Cette partie aborde le signal. Le domaine étant très vaste et les concepts très nombreux, la rédaction du document sera dirigée par la mise en pratique que nous en ferons dans la quatrième partie qui teste notre implémentation de Linda

La fréquence f est l'inverse de la période T. Elle est exprimée en Hertz (avec comme unité la seconde). Elle s'exprime donc par la formule : $f = 1/T$. Avant de pouvoir traiter l'audio avec l'outil informatique, il faut impérativement le transformer. Cette opération s'appelle la numérisation. " La numérisation d'un signal est l'opération qui consiste à faire passer un signal de la représentation dans le domaine du temps et des amplitudes continus au domaine des temps et des amplitudes discrets." [Cottet, 1997, p. 16].

Les deux étapes principales de la numérisation sont l'**échantillonnage** et la **quantification**. Nous ne parlerons pas ici de la dernière opération qui est la restitution : passage du numérique à l'analogique.

3.2 L'échantillonnage et la quantification

3.2.1 L'échantillonnage

L'échantillonnage d'un signal consiste à utiliser un dispositif qui va enregistrer à une certaine fréquence le signal. Nous allons obtenir à un instant t la valeur du signal. Nous allons donc mesurer le signal à des instants $t_0, ..., t_n$ tel que $t_{i+1} - t_i = T_e$. La fréquence d'un échantillonnage est définie par la formule $1/T_e$. Elle doit impérativement être au minimum le double de la fréquence la plus haute que l'on veut échantillonner. Par exemple, si un signal sinusoïdal possède une période de 10 Hz et si notre échantillonnage de même fréquence est en phase, nous n'enregistrerons aucune valeur pour le signal par unité de mesure. Sachant que par unité de mesure le signal passe par une amplitude[1] maximale, nous devons dès lors prendre la mesure à ce moment qui se situe dans notre exemple à 1/2 unité de mesure. Nous devons donc diviser par 2 notre période d'échantillonnage de 10 Hz, soit 20 Hz. Nous

[1] Ampleur de l'oscillation d'une onde par rapport à sa valeur moyenne.

pouvons étendre ce raisonnement pour toutes les formes de signaux.

3.2.2 La quantification

La quantification d'un signal consiste à exprimer sous forme numérique l'amplitude du signal. Pour ce faire, nous allons enregistrer cette information sous forme de bit. Le nombre de bits permet un nombre limité de valeurs. Cette différence entre le signal continu et son expressivité binaire s'appelle le bruit de quantification. Il se remarque par exemple par l'apparition de scratches. Une quantification trop faible limite aussi la qualité des traitements numériques. A partir de 24 bits, le bruit de quantification devient négligeable. De plus, une quantification sur 24 bits permet une meilleure qualité des traitements numériques.

3.3 Introduction au produit de convolution

Imaginons que la lentille d'un télescope mis sur orbite autour de la terre nous envoit une image d'un trait fin en créant de part et d'autre de ce trait un halo de lumière centré autour du trait orignal. Ce halo peut être décrit par ce que l'on appelle une fonction d'étalement ou une fonction de réponse notée $h(x)$. Cette fonction est aussi appelée dans la littérature anglaise line spread function (LSF) ou point spread function (PSF).

Réalité	TéléscopeCloug
Trait en x_1	halo centré en $x_1 : h(x - x_1)$
Intensité en f_1	$f_1.h(x - x_1)$

De même, si nous avions un autre trait x_2 et en supposant que notre télescope (qui est en fait ici un système de transmission) se comporte de la même manière sur toute sa surface de vision et reste constant dans sa réponse quelque soit le moment d'observation, nous aurions :

Réalité	TéléscopeCloug
Trait en x_2	halo centré en $x_2 : h(x - x_2)$
Intensité en f_2	$f_2.h(x - x_2)$

En supposant que nous ayons plusieurs traits x_n d'intensité respective f_n, notre signal de sortie serait la sommation des réponses individuelles avec une possibilité de chevauchement et s'écrirait :

$$S(x) = \sum_{i=1}^{n} f_i h(x - x_i) \tag{3.1}$$

Nous pouvons étendre cette formule si nous considérons que le signal d'entrée est une fonction continue de x et l'écrire sous la forme d'une intégrale :

$$S(x) = \int_{-\infty}^{+\infty} f(x)(u - x)dx \tag{3.2}$$

Cette intégrale est appelée produit de convolution et est notée $f(x) * h(x)$.

3.4 Les filtres

3.4.1 Définition des filtres

Les filtres sont définis comme des systèmes de transmission linéaires [2] continus[3] et stationnaires[4]. Ces filtres sont des systèmes de convolution.

3.4.2 Les filtres à réponse impulsionnelle finie

Les filtres numériques à réponse impulsionnelle finie (filtre RIF) sont des filtres linéaires stables. Ils sont définis par "une équation selon laquelle un nombre de sorties, représentant un échantillon du signal filtré, est obtenu par une sommation pondérée d'un ensemble fini d'entrées, représentant les échantillons du signal à filtrer." [Bellanger, 2002, p. 138]

Un filtre RIF peut être écrit par l'équation aux différences suivante[5] :

$$y[k] = b_0 \cdot x[k] + b_1 \cdot x[k-1] + b_2 \cdot x[k-2] + ... + b_N \cdot x[k-N] \tag{3.3}$$

De manière plus compacte nous pouvons l'écrire sous forme :

$$y[k] = \sum_{i=0}^{N} b_i \cdot x[k-i] \tag{3.4}$$

Le lecteur trouvera en annexe (cfr A.0.2 page 118) un programme en Python d'un filtre à réponse impulsionnelle finie.

Les filtres à réponse impulsionnelle infinie

"Les filtres numériques à réponse impulsionnelle infinie (ou filtres RII) sont des systèmes linéaires discrets invariant dans le temps et dont le fonctionnement est régi par une équation de convolution portant sur une infinité de termes." [Bellanger, 2002, p. 220]

Un filtre RII peut être écrit par l'équation aux différences suivante [6] :

$$y[k] = (b_0 \cdot x[k] + b_1 \cdot x[k-1] + .. + b_N \cdot x[k-N]) - (a_1 \cdot y[k-1] + a_2 \cdot y[k-2] + ... + a_M \cdot y[k-M]) \tag{3.5}$$

[2] En considérant $s_1(t)$ réponse de $e_1(t)$ et $s_2(t)$ réponse de $e_2(t)$, le système est dit *linéaire* si : $ae_1(t) + be_2(t)$ a pour réponse $as_1(t) + bs_2(t)$

[3] Soit $s_n(t)$ la suite des réponses de $e_n(t)$, le système est dit *continu* si nous respectons la propriété suivante : $\lim_{n\to\infty}(s_n(t))$ est identique à la réponse du signal $\lim_{e\to\infty}(e_n(t))$

[4] Un système est dit stationnaire si son comportement est indépendant de l'origine du temps. Par conséquent : $e(t)$ a pour réponse $s(t) \Rightarrow e(t-\tau)$ a pour réponse $s(t-\tau)$

[5] Pour les coefficients nous avons suivi la même nomenclature que dans Scipy.

[6] Pour les coefficients nous avons suivi la même nomenclature que dans Scipy.

De manière plus compacte nous pouvons l'écrire sous la forme :

$$y[k] = \sum_{i=0}^{N} b_i \cdot x[k-i] - \sum_{j=1}^{N} a_j \cdot y[k-j] \tag{3.6}$$

Lorsque l'on donne les coefficients a, il faut bien vérifier que c'est cette formule sous forme soustractive qui est utilisée et non celle sous forme additive.

Le lecteur trouvera en annexe (cfr A.0.3 page 119) un programme en Python d'un filtre à réponse impulsionnelle infinie.

3.4.3 Domaine temporel ou domaine fréquentiel

Produit de convolution

Après la numérisation du signal, nous obtenons une suite temporelle d'échantillons. Le traitement peut se faire de manière temporelle en utilisant le produit de convolution.

Toutefois, nous pouvons aussi souhaiter travailler dans le domaine fréquentiel. Pour cela, nous pouvons utiliser une variante de la transformée de Fourier : la transformée de Fourier discrète.

Transformée de Fourier et transformée de Fourier discrète

Du nom de son inventeur Jean-Baptiste Joseph Fourier [7], le théorème de Fourier s'applique à une fonction périodique. Il permet de décomposer un **signal continu** et **périodique** en ses composantes fréquentielles. Le théorème de Fourier dans sa forme moderne s'écrit d'après [Cottet, 1997, p. 19] :

$$s(t) = a_0 + \sum_{n=1}^{\infty} (a_n cos 2\pi F_0 nt + sin 2\pi F_0 nt) \tag{3.7}$$

avec $s(t)$ une fonction de t périodique, de période $T_0 (= 1/F_0)$. a_n et b_n sont les coefficients de la série de Fourier et a_0 appelé valeur moyenne ou composante continue.

Ce théorème de Fourier peut s'étendre à des fonctions non périodiques. La période est alors infinie ($T_O \rightarrow \infty$) et la fréquence F_0 tend vers zéro. La transformée de Fourier $s(f)$, notée $S(f)$, s'écrit d'après [Cottet, 1997, p. 24] :

$$S(f) = \int_{-\infty}^{+\infty} s(t) e^{(-j2\pi ft)} dt \tag{3.8}$$

Mais comme nous travaillons avec des **signaux discrets** et **non périodiques**, nous utiliserons la transformée de Fourier discrète (TFD ou DFT en anglais pour Discrete Fourier Transform). Celle-ci est en fait l'adaptation de la transformée de Fourier. D'une part, le signal $s(t)$ est remplacé par des nombres

[7] Mathématicien et physicien français (1768-1830).

$s(nT)$ qui représentent un échantillonnage de ce signal et, d'autre part, l'ensemble des nombres sur lequel portent les calculs à une valeur finie N est limité. [Bellanger, 2002, p. 66]

Nous avons alors l'expression :

$$S(f) = \sum_{n=0}^{N-1} s(nT)e^{(-j2\pi fnT)} \tag{3.9}$$

Les calculs sont effectués pour un nombre limité de valeurs de la fréquence f, qui est un multiple de pas de fréquence Δf. On choisit pour Δf la valeur $1/NT$. [Bellanger, 2002, p. 66]

Nous obtenons alors la formule :

$$S(m\Delta f) = \sum_{n=0}^{N-1} s(nT)e^{(-j2\pi nm\Delta fT)} \tag{3.10}$$

Puisque $\Delta f = \frac{1}{NT}$ et que $Fe = \frac{1}{Te}$ l'équation devient

$$S(m\frac{Fe}{N}) = \sum_{n=0}^{N-1} s(nT)e^{(-j2\pi nm\frac{Fe}{N}T)} \tag{3.11}$$

Comme nous itérons sur les échantillons, nous pouvons poser que s(nT) devient s_k et que k désigne l'indice de l'échantillon. L'équation devient alors :

$$S(m\frac{Fe}{N}) = \sum_{k=0}^{N-1} s_k e^{(-j2\pi km\frac{Fe}{N}T)} \tag{3.12}$$

Si nous remplaçons dans le second membre Fe par $\frac{1}{T}$, nous obtenons après simplification l'équation définitive telle que citée par [Cottet, 1997, p. 167] et dont voici la définition : la Transformée de Fourier Discrète F_d d'un signal défini par N échantillons est la suite de N termes définie par :

$$F_d\{s_k\} = S_m = \sum_{k=0}^{N-1} s_k e^{(-j2\pi\frac{km}{N})} \tag{3.13}$$

Ce qui veut dire que si nous disposons de 1024 échantillons successifs d'un signal dont la fréquence d'échantillonnage est de 44100 Hz, nous cherchons les fréquences qui composent S_m. Pour m = 3, nous cherchons en fait le poids de la fréquence $m\frac{Fe}{N}$ soit $129 = 3 \cdot \frac{44100}{1024}$. La transformée de Fourier Discrète nous fournira non seulement le poids de la fréquence 129 mais aussi sa phase. On l'aura compris le résultat est un nombre complexe.

"La transformée de Fourier discrète réalise donc la correspondance entre deux suites de N termes", [Cottet, 1997, p. 167] l'un dans le domaine temporel et l'autre dans le domaine spectral. Pour le domaine temporel, nous avons un signal échantillonné limité. Pour le domaine spectral, nous avons un spectre échantillonné limité.

Transformée de Fourier discrète inverse

La transformée de Fourier discrète inverse permet quant à elle de passer du domaine fréquentiel au domaine temporel. Voici sa formule selon [Cottet, 1997] :

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{(-i2\pi \frac{kn}{N})} \tag{3.14}$$

Effet du fenêtrage temporel

Le calcul de la transformée de Fourier d'un signal continu à l'aide d'un ordinateur implique sa discrétisation (échantillonnage) ainsi qu'un troncage temporel. Le troncage temporel s'effectue lorsque l'on passe du domaine $-\infty$ à $+\infty$ à un nombre fini d'échantillons k. Ce fenêtrage temporel induit une déformation du spectre. Plusieurs auteurs proposent de modifier cette fenêtre naturelle en appliquant une formule pour produire des nouveaux échantillons. Voici d'après [Cottet, 1997, p. 184], les principales fenêtres. Celles-ci sont présentées par ordre croissant d'efficacité au regard de la minimisation de la déformation (Smoothing Windows) :

Fenêtre de Hanning [8]
$0,5(1 - cos(\frac{2\pi t}{\tau})$

Fenêtre de Hamming :
$0,54 - 0,46cos(\frac{2\pi t}{\tau})$

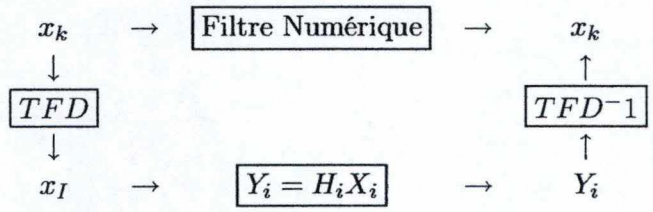
Fenêtre de Blackman
$0,42 - 0,5cos(\frac{2\pi t}{\tau}) + 0,08cos(\frac{4\pi t}{\tau})$

Fenêtre de Blackman exacte
$0,42659071 - 0,49656062cos(\frac{2\pi t}{\tau}) + 0,07684867cos(\frac{4\pi t}{\tau})$

Fenêtre de Blackman-Harris
$0,42323 - 0,49755cos(\frac{2\pi t}{\tau}) + 0,07922cos(\frac{4\pi t}{\tau})$

Relation entre produit de convolution et transformée de Fourier discrète

Le tableau 3.1 extrait [Cottet, 1997, p. 197] schématise la relation entre produit de convolution et transformée de Fourier discrète :



TAB. 3.1 – Réalisation d'un filtre numérique par transformée de Fourier discrète avec H_i la fonction de tranfert discrète du filtre.

Chapitre 4

Le son

4.1 Introduction

Dans cette partie, nous allons aborder de manière pratique l'audio. Un fichier audio peut être composé de multitudes de fréquences. La fréquence "étant le nombre de fois qu'un phénomène périodique se produit par unité de temps"[Wikipedia, 2009e].

Après la **numérisation du signal**, il faut stocker l'information pour permettre sa réutilisation. Différents choix s'offrent alors : soit on stocke toute l'information, soit on tente de ne garder que la partie que l'on juge intéressante. Le stockage peut alors se faire soit sous forme compressée soit sous forme brute. Il existe une diversité de formats de fichier qui rencontrent ces dernières préoccupations.

4.2 Le format idéal

Parmi la diversité des formats proposés tant au niveau du stockage que de la diffusion, les contraintes imposées par un traitement massif de l'information nous obligent à prendre en considération la vitesse que peuvent proposer ces formats. En effet, l'accès à l'information est souvent très pénalisante si l'on considère la vitesse entre l'unité centrale de l'ordinateur et les périphériques de stockage locaux ou distants. D'autre part, si notre choix se porte sur un format compressé, nous devons décider si nous pouvons admettre ou non une perte de l'information.

Le choix sera aussi guidé par le rapport qui existe entre la vitesse d'encodage et la vitesse de décodage du format. " Un encodage est dit asymétrique lorsque le temps nécessaire à sa réalisation est entièrement décorelé avec le temps nécessaire du décodage. " [Wikipedia, 2009j] En effet, la plupart des formats sont dits asymétriques (voir l'info) car souvent ils mettent plus de temps pour encoder l'information que pour la décoder. Vous trouverez sur ce site [Kurtnoise, 2004] un comparatif entre différents formats sans perte (lossless en anglais).

4.3 Notre choix

Parmi la multitude de formats, notre choix s'est arrêté sur deux formats : le wav de Microsoft et le flac.

4.4 Le format wav

Le format wav a été défini conjointement par IBM et par Microsoft. Il hérite des spécifications du format RIFF. Ce dernier est un standard pour l'échange de données : RIFF étant l'abréviation pour Ressource Interchange Files. On peut considérer un fichier de type RIFF comme un container pouvant contenir différents types de contenu comme par exemple de l'audio, de la vidéo ou des donnés Midi.[Scott, 2003] Le format wav est donc lui-même un sous-container audio de type RIFF. Il peut accueillir divers formats audio tels que le **mp3** ou le **pcm**[1].

C'est ce format qui est le plus souvent utilisé. Comme ce format est sans perte, on considère à tort que tous les fichiers wav sont sans perte. [Wikipedia, 2009i]

Le lecteur trouvera en annexe les spécification du format wav. (cfr B.1 page 122).

Des outils sont aussi présentés en annexe pour recueillir des informations sur le fichiers wav. (cfr B.2 page 122).

4.5 Le format flac

Flac est un codec [2] audio live. Ce dernier est un format compressé sans perte. Nous ne détaillerons pas son format. Le lecteur pourra trouver plus de détails dans [Wikipedia, 2009d].

4.6 Wav versus Flac

Nous avons testé la vitesse d'encodage-décodage des deux formats. Le lecteur intéressé touvera en annexe un comparatif (cfr B.7.1 page 124) entre le format wav et le format flac qui reprend des tests de lecture et d'écriture en utilisant soit un programme compilé (sox) soit un script en Python.

L'occupation de l'espace disque a été aussi testée (cfr B.7.4 page 129). Malgré un net avantage du format Flac quant à l'occupation disque , nous avons décidé de privilégier la vitesse d'encodage-décodage et donc de choisir le **format wav** d'autant plus qu'il possède les avantages suivants :

- c'est un format bien documenté et simple à comprendre ;
- on le rencontre souvent sans compression. Les données sont ainsi facilement accessibles sur le support de stockage puisqu'elles y sont écrites de manière brute, octet par octet ;
- il n'y a pas de perte de signal[3] ;
- il est libre d'utilisation (contrairement au mp3 par exemple) ;
- comme c'est un format historique, il est presque toujours implémenté dans les bibliothèques de traitement du signal ;
- il ne provoque pas de perte.

[1] Pulse Code Modulation

[2] Codec vient de **com**pression-**déc**ompression.

[3] Dans la plupart des cas.

Chapitre 5

L'image

5.1 Introduction

Comme pour la partie consacrée au son, nous allons orienter cette partie par l'approche pratique. Le traitement numérique des images regroupe en fait différents domaines. Cela va de l'acquisition à son analyse sémantique. Il existe différentes taxonomies. Remarquons que les frontières entre les différents domaines sont parfois floues. La reconnaissance des formes, par exemple, peut être de bas niveau comme de haut niveau. Quant à nous, nous retiendrons les trois catégories que proposent Gonzalez et Woods dans [Gonzalez et Woods, 2002, p.2]. Ces deux auteurs définissent le traitement de l'image dans un continuum divisé en trois niveaux :

- le bas niveau : c'est l'utilisation d'opérations comme par exemple la réduction du bruit ou l'augmentation du contraste. L'input et l'output sont des images ;
- le niveau moyen : c'est par exemple toutes les opérations de segmentation (division de l'image en zones). L'input est une image alors que l'output sont les attributs de l'image ;
- le haut niveau : ce sont les opérations qui donnent du sens aux informations que l'on extrait, Ce sont les opérations cognitives qui sont associées avec la vision.

5.2 Échantillonnage et quantification

La numérisation d'une image fait intervenir deux étapes : l'échantillonnage et la quantification.

5.2.1 L'échantillonnage

L'échantillonnage consiste à diviser l'image en lui appliquant une grille à deux dimensions pour obtenir une matrice à deux dimensions. Cette grille peut être composée d'éléments de toute forme, notamment les formes rectangulaires ou elliptiques que l'on rencontre dans les arts graphiques. Dans le monde informatique, la grille choisie est généralement carrée. OpenGl qui est une bibliothèque graphique choisit cette option. La plus petite unité ainsi obtenue se nomme le pixel (contraction de l'anglais picture element) [Wikipedia, 2009f].

5.2.2 La quantification

La quantification consiste à transformer un signal analogique (variation continue d'un signal physique) en un signal à valeurs discrètes. La quantification traduit l'intensité de chaque pixel dans l'espace colorimétrique (cfr infra) choisi en le codant sur n bits.

Les images numériques les plus connues sont de trois types :

- Binaire $I(x, y) \in [0, 1]$
- Niveau de gris $I(x, y) \in [0, 256]$ [1]
- Couleur $I_R(x, y)I_V(x, y)I_B(x, y) \in [0, 255]$ [2]

5.3 Les espaces colorimétriques

5.3.1 Les espaces colorimétriques normalisés

Un espace colororimétrique normalisé sert de référence et de standard. Il permet d'exprimer les valeurs d'une couleur qu'il peut représenter. Il en existe une très grande variété.

L'espace CIELab, par exemple, est un espace colorimétrique qui se base sur la perception de l'oeil humain.[Adobe, 1992, page 112] Il est donc adapté à la lumière visible par l'homme. Le rvb, quant à lui, est associé à la vidéo. sRVB définit l'espace colorimétrique du moniteur standard.

5.3.2 Choix d'un espace normalisé

Le choix adéquat de l'espace colorimétrique doit être opéré en fonction de ses qualités intrinsèques, du type d'usage et du traitement que l'on compte faire. Le traitement numérique des images se heurte aux propriétés mathématiques de l'espace colorimétrique. Un espace colorimétrique linéaire simplifiera grandement les calculs.

Voici quelques critères qui, selon [Adobe, 1992, page 112], lui permettent de recommander l'espace CIELab :

- colorimétrie proche de la vision humaine ;
- indépendance matérielle ;
- couvrant tout le gamut ;
- efficient en utilisant un espace uniforme ;
- standardisation indépendante de toute société commerciale ;
- séparation de la chromiance et de la luminance ;
- plus de compression sans réelle perte.

5.3.3 L'espaces colorimétrique

Un espace colorimétrique exprime la capacité que possède une sortie à reproduire un espace de couleurs. On peut imaginer par exemple que l'ensemble des couleurs qu'une imprimante peut reproduire est différent de ce que peut reproduire un écran vidéo.

[1] En cas de codage sur 8 bits et [0,65536] pour un codage sur 16 bits.

[2] Les indices représentent le système de codage des couleurs RVB.

5.3.4 Le profil icc

Le profil icc établit une correspondance entre l'espace colorimétrique lié à un périphérique et un espace colorimétrique normalisé. Cette information peut être importante dans le cas de traitement numérique car deux couleurs identiques peuvent être enregistrées avec des valeurs différentes en fonction du profil icc choisi. Le profil icc permet donc de s'assurer de la cohérence des informations couleurs.

5.3.5 Le dématriçage

Actuellement on rencontre deux types de capteur de lumière : les CCD (Charged Coupled Device) et les CMOS (Complementary Metal Oxyde Semicondutor). Ces micro-circuits électriques constituent ce que l'on appelle les photosites. Les photosites permettent d'enregistrer des tensions en fonction de la lumière qu'ils reçoivent. Malheureusement, ils ne peuvent enregistrer qu'une longueur d'onde dans la majorité des cas.

L'organisation de ces photosites est faite en grille ou en mosaïque. La grille peut être organisée en lignes et en colonnes ou en quinconce.

L'image brute obtenue est soit totale, soit partielle.

Dans cette première catégorie, on retrouve la technologie triccd et FOVEON.

Grâce à des filtres dichroïques, la lumière est décomposée en trois faisceaux qui frappent trois surfaces sensibles correspondant respectivement aux trois couleurs primaires (RVB).

La technologie FOVEON organise ses capteurs en couches horizontales superposées (bleue, verte et rouge respectivement de haut en bas).

Ces deux technologies sont actuellement les plus répandues.

Dans le type d'image partielle, les photosites sont spécialisés dans une longueur d'onde et sont organisés selon une matrice. La matrice la plus commune est la matrice dite de Bayer. Son organisation est la suivante : chaque photosite enregistre une valeur sur trois, les deux autres valeurs étant interpolées. Cette opération s'appelle le dématriçage.

5.4 Les formats de fichier image

5.4.1 Choix d'un format de fichier image

Le choix d'un format dépend de l'objectif que l'on poursuit. Si l'image est simplement un objet d'illustration, le format choisi sera probablement un équilibre entre son poids et la qualité de rendu que l'on souhaite obtenir.

Dans ce domaine, il existe des structures de fichier qui remplissent des rôles généralistes ou spécifiques. Certains formats sont utilisés pour la diffusion, d'autres pour le stockage. On rencontre dans certains domaines des formats spécifiques : dans l'imagerie médicale ou dans la recherche scientifique.

5.4.2 Les critères objectifs de choix

Il existe des centaines de formats de fichier image. Beaucoup ne sont plus guère utilisés.

Dans le cas d'un traitement numérique, les contraintes peuvent notamment être dictées par :

- le type de traitement ;
- l'usage du fichier : conservation de l'information sur un support ou échange ;
- le type et la forme des informations que le fichier permet de stocker ;
- sa standardisation.

Certains formats ne se rencontrent que dans certains domaines. Citons par exemple le format dicom dans le domaine médical qui permet d'enregistrer plusieurs clichés et ainsi regrouper le dossier du patient.

Pour notre part, sur base de ces critères, nous avons choisi les formats suivants :

- pour les données brutes : le raw et le dng ;
- pour l'échange : le tiff et le jpeg ;
- pour le traitement intermédiaire : le ppm ou le codage en base64.

5.4.3 Raw

Les données ainsi numérisées peuvent être stockées sous une forme brute. Mais, de la même manière qu'il y a un développement dans le processus argentique, on peut aussi par analogie employer le même terme. On peut ainsi se contenter d'enregistrer les informations sous une forme brute dans un format qui est appelé format raw.

Le format raw n'est pas à proprement parler un format image. Il reprend trois types d'information :

- les densités brutes qui ont été captées par l'appareil photographique ;
- les paramètres de l'appareil tels que :
  - le temps de pose ;
  - l'ouverture du diaphragme ;
  - la sensibilité de l'appareil ;
- les valeurs taguées pour le développement telles que la balance des blancs.

Son principal avantage est qu'il peut être développé. C'est-à-dire que les informations qu'il enferme permettent à l'utilisateur de sélectionner ou de modifier les données enregistrées par l'appareil photographique en fonction de l'usage ou du résultat qu'il souhaite obtenir. Un fichier raw peut ainsi donner plusieurs interprétations que l'on appelle tirage.

Malheureusement, chaque constructeur a proposé son format propriétaire rendant l'exploitation de ces négatifs numériques peu interopérables.

Adobe a proposé un format ouvert et documenté que l'on peut librement utiliser : il s'agit du format DNG.

Le lecteur pourra trouver les informations dans l'excellent ouvrage de Gilbert [Gilbert, 2009].

5.4.4 Tiff

"Le Tag(ged) Image File Format (généralement abrégé TIFF) est un format de fichier pour image numérique. Adobe en est le dépositaire et le propriétaire initial." [Wikipedia, 2009g]

Ce format est compatible avec une multitude de logiciels. En voici les caractéristiques :

- reconnu par beaucoup de logiciels ;
- standard ;
- peut contenir des méta-données standards ou spécifiques ;
- peut contenir plusieurs images.

5.4.5 Jpeg

Ce format doit sa popularité au Web. C'est un format compressé. Lors de la compression, on peut choisir le taux de qualité que l'on souhaite obtenir. Plus le taux de qualité sera faible, plus la compression sera forte. Voici à titre indicatif la taille d'un fichier jpeg :

JPEG	
100	2,95 Mo
80	0,51 Mo
60	0,36 Mo
40	0,30 Mo

Remarquons que le gain n'est pas proportionnel à la qualité.

Cette compression utilise la tranformée de Fourrier discrète. (cfr 3.4.3 page 37)

5.4.6 Comparaison des fichiers

Format	Qualité	Poids
RAW	-	9 Mo
TIFF	-	17,7 Mo
JPEG	100	5,08 Mo
JPEG	80	1,15 Mo
JPEG	60	0,79 Mo
JPEG	40	0,63 Mo

Il est intéressant de noter que le format raw offre le meilleur rapport qualité/poids. Il va même jusqu'à disqualifier le format tiff qui ne contient pas plus de détails mais qui pèse presque le double. Cela est expliqué par le fait que le format raw ne contient qu'une information sur trois relative à la quantité de chaque couleur RVB. (cfr : 5.3.5 45)

Plus les images possèdent de détails, plus elles seront difficilement compressibles.

Deuxième partie

Analyse de l'existant

Introduction

Dans cette partie nous nous intéresserons à quatre implémentations que nous avons trouvées lors de nos recherches sur le Net. Nous avons donné la priorité à des implémentations qui offraient des clients ou des apis en Python parce que nous avons fait le choix de ce langage pour explorer le domaine traité par notre mémoire.

Nous nous sommes aussi tourné vers des implémentations dont le code-source était disponible, cela étant justifié par deux objectifs principaux :

- analyser facilement l'implémentation et pouvoir vérifier par nous-mêmes le discours de l'implémenteur ;
- utiliser l'implémentation tout en l'adaptant si nous le souhaitions.

Notre recherche s'est focalisée sur deux sources importantes de logiciel libre que sont sourceforge (`http://www.sourceforge.net`) et freshmeat (`http://www.freshmeat.net`).
Deux sites qui sont connus et reconnus par la communauté open-source.

Nous avons aussi complété notre recherche via des méta-moteurs spécialisés mais ces recherches n'ont rien apporté de neuf.

Les implémentations que nous allons présenter possèdent des degrés de maturité différents. Pybrenda est la moins mature tandis que Pylinda est le projet le plus abouti. Quant à Linuxtuples, il est situé entre les deux.

Nous parlerons brièvement aussi du quatrième mousquetaire : Networkspaces. Cette dernière implémentation n'a été portée à notre connaissance que très tardivement. En effet, pour des raisons que nous n'arrivons pas à comprendre, bien que ses sources soient hébergées sur Sourceforge, cette implémentation n'est référencée sous aucun mot-clé tel que spaces, linda ou tuples. Nous ne devons sa découverte que parce que nous voulions avoir un exemplaire de *How to write parrallel programs, A first course* [Carriero et Gelernter, 1990]. Une recherche sur Internet nous conduisit alors sur un site qui mettait à disposition une copie de l'ouvrage. Enthousiasmé par notre découverte, nous avons immédiatement téléchargé le dit exemplaire sans prêter attention à l'adresse. C'est ensuite en voulant vérifier l'origine du document que nous avons découvert que ce site, qui nous était bien connu, mettait aussi à disposition Networkspaces qui est une implémentation de Linda en Python. Cela nous a troublé puisque ce site n'est autre que `www.lindaspaces.com` dont la relation avec les fondateurs de Linda est bien connue.

Par honnêteté intellectuelle, nous ne pouvions passer sous silence cette découverte même si elle nous embarrassait. Nous ne pouvions malheureusement lui donner la place qu'elle mérite probablement dans notre analyse. Le manque de temps nous imposait de renoncer à son exploration approfondie. Toutefois, nous présenterons son implémentation et son architecture de manière succinte.

Signalons qu'il existe d'autres projets dans d'autres langages dont la qualité est équivalente ou supérieure aux quatre implémentations que nous avons analysées. Ces implémentations sont utilisées de manière intensive en production. Sun et IBM, par exemple, proposent une implémentation des

principes de Linda. Mais notre objectif était bien de trouver des implémentations en Python et non la meilleure implémentation de Linda.

Chapitre 6

Présentation des quatre implémentations

6.1 Pybrenda

Pybrenda est une implémentation que l'on pouvait trouver sur le site : `http://www.snurgle.org/~pybrenda`. Ce site est archivé à l'adresse `http://web.archive.org/web/*/http://www.snurgle.org/~pybrenda`.

Cette implémentation est référencée sur le site officiel de Python à l'adresse `http://wiki.python.org/moin/PyBrenda`. Les sources sont disponibles à l'adresse `http://xml.coverpages.org/PyBrenda041-tar.gz`.

Notons que Pybrenda figure aussi à l'adresse http ://xml.coverpages.org/tupleSpaces.html où nous trouvons un inventaire des solutions Linda. Elle y est présente notamment à côté de Tspaces (implémentation fournie par IBM). Ajoutons qu'elle est aussi référencée sur le site `http://www.c2.com/cgi/wiki?TupleSpace`. Nous avons constaté au fil de nos recherches sur le Web que cette implémentation assez ancienne était encore référencée.

L'auteur est Milton L. Hankins.

Pybrenda est développée depuis 1999. Nous avons utilisé la seule version qui est encore accessible, c'est-à-dire la version 0.4.1.

Cette implémentation se trouve sous licence Artistique.

Elles est destinée aux machines de type Unix mais ne fonctionne pas sous Solaris. Elle fonctionne correctement sous Linux précise l'auteur.

6.2 Linuxtuples

Cette implémentation est disponible à l'adresse suivante : `http://linuxtuples.sourceforge.net/`.

Elle est sous licence bsd.

Ce projet a été enregistré le 3 août 2003 et la dernière mise à jour a été faite le 20 septembre 2007. Il est présenté comme étant une implémentation des tupleSpaces selon David Gelernter.

La popularité de ce projet est assez faible puisqu'il ne recueille que 134 download entre septembre

2008 et août 2009.

Linuxtuples est destinée aux machines dont le système d'exploitation est de type Unix* et spécialement Linux.

L'auteur est Will Ware sans autre précision. Son email est : wware@alum.mit.edu.

Sur la page Web, nous trouvons une très brève introduction à Linda.

6.3 Pylinda

Cette implementation a été développée par Andrew Wilkinson. Le projet a d'abord été hébergé à l'Université de York à l'adresse `http://www-users.cs.york.ac.uk/~aw/pylinda` pour ensuite migrer vers le site de `code.google.com`. L'annonce a été faite sur la liste de diffusion le 22 septembre 2006.

Le site actuel de référence est : `http://code.google.com/p/pylinda/`.

Une version de développement est téléchargeable via le programme de versionning subversion[1] à l'adresse `http://pylinda.googlecode.com/svn/trunk/pylinda-read-only`.

Nous avons utilisé la version 0.6 du projet qui a été rendue publique le 6 février 2006.

Cette implémentation disposait d'une certaine audience sur le Net. En effet, une petite communauté s'est créée autour de cette version et disposait d'un groupe d'échange à l'adresse `http://groups.google.com/group/pylinda/`. Ce groupe n'est plus actif aujourd'hui.

6.4 NetworkSpaces

Cette implémentation est développée par Scientific Computing Associates Inc qui a acquis une certaine réputation en proposant différentes implémentations du modèle Linda.

Le site de référence est `www.lindaspaces.com`. Les sources sont hébergées sur Sourceforge et sont accessibles à l'adresse `nws-py.sourceforge.net/`.

Le logiciel est sous licence gpl version 2 ou ultérieure.

Cette implémentation se veut complète et destinée au monde scientifique puisqu'elle permet dès l'origine une utilisation de Matlab (outils de calcul scientifique) et r (outils statistiques).

Elle n'est pas limitée au monde Unix.

Ce projet en est à sa version 1.6.3 pour la version client et à la version 1.5.2 pour le serveur Python au moment de la rédaction de notre mémoire.

Ce projet peut être plus décrit comme étant un framework permettant de coordonner des programmes écrits en Python, en Matlab ou en R.

Comme nous le verrons, cette implémentation est une relecture du modèle de base et plus particulièrement de la notion de tuple. Cette relecture nous a paru intéressante à présenter.

NetworkSpaces est le plus complet des quatre projets.

[1] svn checkout l'adresse du lien

Chapitre 7

La grille d'analyse

7.1 Architecture logicielle

Pour l'architecture logicielle, nous utilisons une grille d'analyse que nous avons construite selon nos lectures et nos connaissances. Cette grille (voir figure 7.1) est présentée sous la forme d'un Mind-Mapping.

La grille se focalise principalement sur quatre axes qui sont :

- le middleware ;
- l'intégration ;
- le modèle de distribution ;
- la persistance des données.

7.2 Le langage de coordination Linda

Toutes les versions que nous avons choisies implémentent peu ou prou la vision de Linda telle que développée par le Linda Team (`http://www.cs.york.ac.uk/linda/pubs.html`).

Nous envisageons d'abord la présence des **primitives de base**, c'est-à-dire :

- out
- in
- rd
- eval

Pour les **tuples vivants** (eval), nous analysons le paradigme qui est utilisé.

Nous cherchons ensuite à savoir si d'autres primitives sont implémentées telles que :

- inp
- rdp
- copy
- copy-collect

Nous analysons alors la syntaxe et la sémantique de l'ensemble des primitives.

Nous nous intéressons ensuite au nombre de tupleSpaces que l'implémentation supporte.

Pour terminer nous examinons, d'une part, les variables qui sont gérées par l'implémentation et, d'autre part, la syntaxe qui est utilisée pour définir un patron.

7.3 L'espace de tuples

Cette partie envisage deux axes :

- les outils de gestion du tupleSpace ;
- la distribution du TupleSpace entre plusieurs sites.

7.4 La distribution de l'implémentation

Nous nous intéressons ici au packaging. Il nous paraît important de voir comment est organisée la distribution de l'implémentation.

La prise en main sera minutieusement examinée. En effet, la critique la plus courante dans les projets open-source est souvent qu'ils sont très difficilement utilisables : soit par manque de documentation, soit par manque d'organisation, soit tout simplement par manque de préoccupation de l'utilisateur final.

Pour l'analyse nous avons retenu cinq axes :

- l'installation ;
- la prise en main ;
- la description des fichiers ;
- la documentation en général ;
- les exemples d'utilisation fournis.

7.5 Les sources

Dans cette dernière partie nous nous intéressons à l'accès au code et aux apis disponibles.

D'emblée nous signalons que toutes les implémentations que nous avons choisies sont open-source, ce qui signifie au minimum que le code est à disposition pour analyse.

7.5.1 Le code

Le code sera lui examiné sous trois axes qui sont :

- le langage utilisé ;
- les tests unitaires ;
- la documentation proprement dite du code.

7.5.2 L'api client

L'api client sera, quant à elle, étudiée par le prisme du dénombrement des clients qui sont disponibles et de la présence de documentation qui explique comment on peut programmer de nouveaux clients.

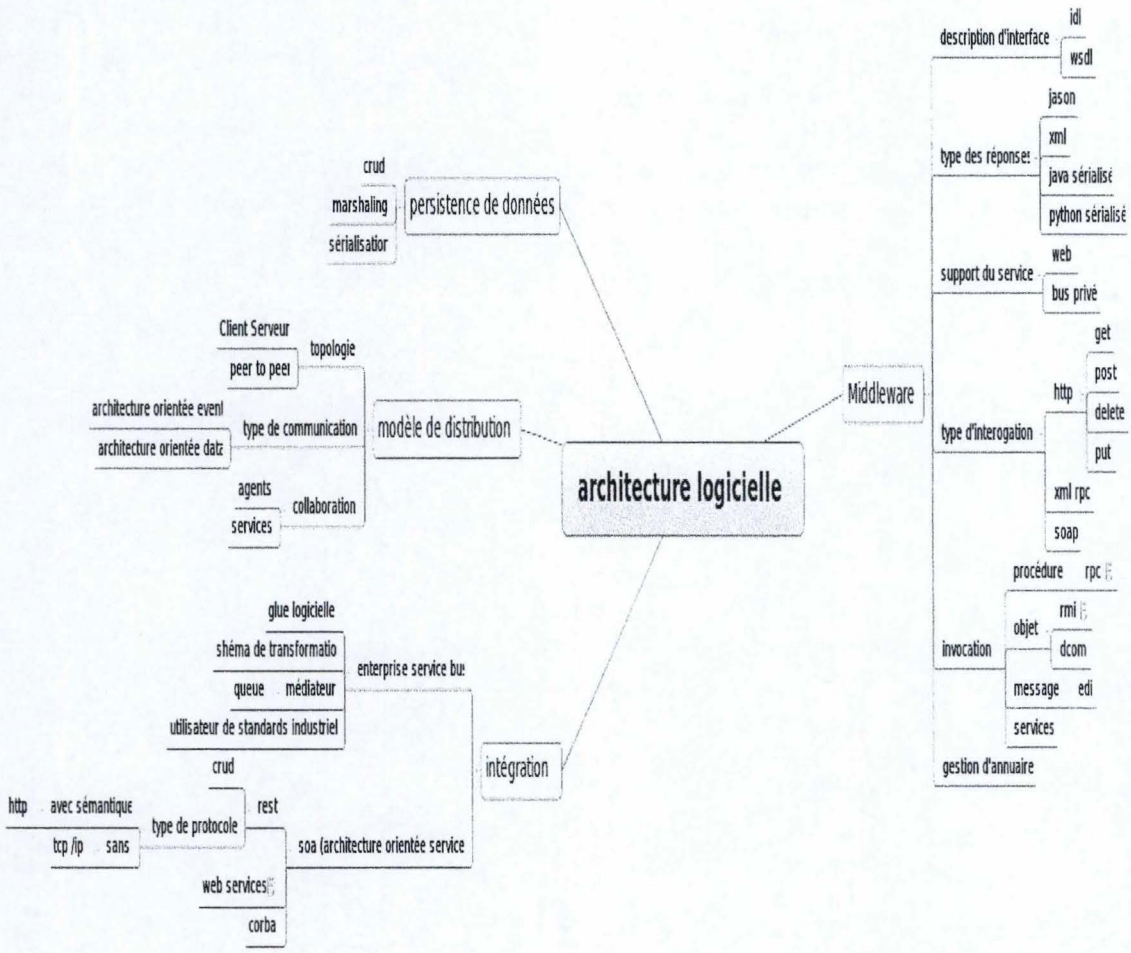


FIG. 7.1 – Grille d'analyse de l'architecture logicielle

Chapitre 8

Analyse Pylinda

8.1 Architecture logicielle

Pylinda utilise le paradigme client-serveur.

Signalons d'emblée que l'architecture client-serveur est particulière. Chaque noeud de calcul accueille en fait un serveur. Si le réseau n'est constitué que d'un seul noeud, le client interroge directement le serveur local qui fait ici office de serveur principal et qui gère donc l'espace de tuples.

Lorsque nous avons plusieurs noeuds, le client doit obligatoirement passer par un serveur local qui interroge le serveur principal. Ce serveur local[1] fait alors office de *proxy* dont les tâches se résument à :

- encoder les requêtes des clients ;
- envoyer des messages au serveur principal ;
- réceptionner les réponses du serveur principal ;
- décoder et envoyer les réponses au client.

Il est donc plus juste de parler de modèle client-proxy-serveur [2].

Cette architecture offre donc un moyen de mettre sur pied un log total des messages envoyés et reçus par le client.

Les deux composants principaux du serveur sont :

- l'espace de tuples ;
- le gestionnaire de messages.

Le **gestionnaire de messages** gère tous les messages venant des clients. Il sert en quelque sorte d'intermédiaire entre l'espace de tuples et les clients.

Les serveurs sont multi-threadés et un thread est créé à chaque nouvelle requête du client.

La communication entre les noeuds utilise les sockets Unix.

Il n'y a pas de persistance des données. En effet, Pylinda implémente un système de *garbage collector* qui, à chaque fois qu'un client meurt, nettoie l'espace de tuples de tous les tuples qui lui appartenaient.

[1] C'est en fait le même code pour les deux types de serveur.

[2] Par abus de langage dans le reste du texte nous parlerons de proxy pour désigner le serveur local qui n'est pas responsable de l'espace de tuples.

Le serveur est capable de détecter les deadlocks. Comme nous le verrons plus tard, cette fonctionnalité est imposée par le fait que le serveur implémente une sémantique particulière pour la primitive **inp**.

8.2 Le langage de coordination Linda

Pylinda implémente les primitives de base à l'exception de eval. Cela est étonnant car c'est une implémentation qui émane de l'université de York dont est issu le *Linda Team*. C'est d'autant plus étonnant que cette primitive est bien référencée dans les primitives de base par le *Linda Team*.

Au lancement du serveur un espace de tuples dont la dénomination est linda.universe permet aux agents de se *donner rendez-vous* dans des espaces privés.

Les espaces de tuples sont des instances de la classe TupleSpaces.[3]

8.2.1 Les primitives de base

- out → instanceOfTupleSpaces._out(tuple)[4] : émet un tuple vers l'espace de tuples ;
- in → instanceOfTupleSpaces._in(tuple) : consomme un tuple de l'espace de tuples [5] ;
- rd → instanceOfTupleSpaces._rd(tuple) : vérifie la présence d'un tuple dans l'espace de tuples[6] ;
- eval → est absent de cette implémentation.

8.2.2 Les primitives supplémentaires

- inp → instanceOfTupleSpaces._inp(tuple) : consomme un tuple de l'espace de tuples[7] ;
- rdp → instanceOfTupleSpaces._rdp(tuple) : vérifie la présente d'un tuple sur l'espace de tuples[8] ;
- copy → instanceOfTupleSpaces.copy()[9] : déplace les tuples d'un espace de tuples vers un autre espace de tuples[10] ;
- copy-collect → instanceOfTupleSpaces.copy_collect(otherInstanceOfTupleSpaces, tuple) [11] : copie les tuples d'un espace de tuples vers un autre espace de tuples[12].

Nous faisons remarquer au lecteur que la primitive **inp** est implémentée selon l'article de Jacob et Wood [Jacob et Wood, 2000].

8.2.3 Le patron

Pylinda utilise nativement la puissance de Python pour gérer les patrons. L'unification se fait sur le type de la variable. Nous avons ici toutes les possibilités qu'offre Python. Pour cela il suffit d'indiquer

[3] Remarquons que tous les arguments des primitives sont bien des tuples et non des arguments.

[4] Remarquons que tous les arguments des primitives sont bien des tuples. Nous retrouvons donc dans le code : instanceOfTupleSpaces._out(("bonjour",2)).

[5] Opération bloquante.

[6] Opération bloquante.

[7] Opération non bloquante.

[8] Opération non bloquante.

[9] Attention au changement malheureux de syntaxe.

[10] Cette opération détruit les tuples d'origine.

[11] Attention au changement malheureux de syntaxe.

[12] Cette opération préserve les tuples d'origine.

un type reconnu par Python dans le tuple pour que l'unification s'opère.

Si nous souhaitons par exemple vérifier la présence d'un tuple dont la première position est un chaîne, la deuxième une liste, la troisième un complexe et dont la dernière position est occupée par le mot "eureka", la requête s'écrira :

- instanceOfTupleSpace._rd((string, list,complex,"eureka")).

Cet exemple illustre bien la puissance offerte par Pylinda pour interroger le serveur principal au moyen des patrons.

8.3 L'espace de tuples

L'implémentation offre une ligne de commande qui permet de surveiller l'espace de tuples. Cette ligne de commande apparaît au lancement du serveur. Avouons que l'interface est plutôt spartiate : seul un prompt sous la forme du caractère "< " apparaît. Un help nous apprend que nous avons à notre disposition :

- **quit** : qui permet d'arrêter le serveur ;
- **list** : qui liste tous les espaces de tuples présents sur le serveur ;
- **inspect** : qui liste les tuples dans l'espace de tuples ;
- **watch** : qui permet d'exécuter une commande régulière.

Cette interface est vraiment minimaliste mais elle a le mérite d'exister. De notre expérience il ressort qu'elle est indispensable pour le débogage. Même si l'affichage est parfois capricieux.

L'implémentation n'offre pas la possibilité de déployer plusieurs serveurs gestionnaires de l'espace de tuples. Cela peut paraître étrange alors qu'il est nécessaire de déployer des serveurs locaux. Mais rappelons que ces serveurs distants sont des proxy.

8.4 La distribution de l'implémentation

8.4.1 L'installation

Après une décompression de l'archive[13] nous obtenons l'arborescence suivante :

```
|-- linda-0.6
|   |-- examples
|   |-- doc
|   |-- monitor
|   |-- Doxyfile
|   |-- PKG-INFO
|   |-- README
|   |-- __init__.py
|   |-- connections.py
|   |-- derived_server.py
|   |-- domain_socket.py
|   |-- kernel.py
|   |-- messages.py
|   |-- options.py
|   |-- profile.py
|   |-- server.py
|   |-- setup.py
```

[13]Rappelons que nous utilisons la version 0.6.

```
|   |-- speed_client.py
|   |-- speed_server.py
|   |-- stats.py
|   |-- tscontainer.py
|   |-- tuplecontainer.py
|   |-- tuplespace.py
|   |-- tuplespacehandler.py
|   |-- utils.py
|   |-- linda_server
```

Andrew Wilkinson introduit très brièvement le modèle Linda dans le fichier README. Dans ce même fichier nous apprenons qu'un simple **"python setup.py install"** suffit pour avoir une implémentation pleinement fonctionnelle.

Le reste du fichier donne des détails sur l'emplacement des fichiers et sur certains problèmes connus de l'implémentation.

8.4.2 La prise en main

Après la lecture du fichier README, il est déjà possible d'écrire son bout de code Linda. La prise en main est vraiment très rapide.

La documentation n'est pas pléthorique mais elle bien équilibrée. Le dossier doc renferme une documentation sous format html. Les rubriques sont explicites et permettent une navigation simple et efficace.

Le tutoriel fait seize lignes. On peut comprendre que l'ensemble de la documentation s'adresse à un public de programmeurs qui est habitué à une documentation austère.

8.5 Les sources

Le code est écrit en Python. Mais nous pouvons remarquer qu'il traîne ça et là quelques fichiers C ou des références à des fichiers C. Le code a été écrit dans ce langage pour accélérer le traitement. Rappelons que Python permet facilement de faire un binding avec un programme en C.

L'architecture proposée par Linda est complexe et la compréhension du code apparaît par vagues successives. Si le code est documenté ça et là, l'auteur aime l'expression *brief* qui apparaît régulièrement.

Nous pouvons regrouper les fichiers[14] en cinq catégories : [15]

- l'api client avec le fichier kernel.py ;
- le serveur avec server.py et connections.py ;
- le gestionnaire de messages avec tuplespacehandler.py ;
- l'organisation des tuples par niveau d'abstraction ascendant :
  - tuplecontainer.py qui définit la classe TupleContainer qui permet d'organiser les tuples. Il s'agit d'un *trie*, c'est-à-dire un arbre numérique ordonné qui est utilisé pour stocker une table associative [Wikipedia, 2009h] ;

[14] La notion de module et de fichier est largement liée en Python : en pratique, un fichier ou un module possèdent la même signification et regroupent au choix des classes, des fonctions ou simplement du code Python.

[15] A la lecture du code, on constate que la séparation est moins tranchée que dans notre présentation. En effet, nous sommes en présence de développement objet et l'auteur réutilise les briques qui servent dans d'autres modules Python.

  - tuplespaces.py qui définit la classe TupleSpace permettant d'intancier un espace de tuples ;
  - tscontainer.py qui définit la classe TupleSpaceContainer qui permet d'instancier l'ensemble des espaces de tuples. Ce dernier est un dictionnaire ;
- les fichiers utilitaires :
  - util.py qui regroupe des fonctions utilitaires en tous genres ;
  - message.py qui reprend la liste de tous les messages supportés par le protocole de Pylinda. Le serveur et les proxy communiquent en effet entre eux par le biais d'un protocole spécifique ;
  - option.py qui est un fichier classique lorsque l'on utilise des options de la ligne de commande ;
  - stats.py qui donne des informations, notamment sur l'occupation de la mémoire.

Andrew Wilkinson a apporté un grand soin à l'optimisation de la recherche des tuples. L'utilisation d'arbres numériques ordonnés avec des dictionnaires (données associatives) permet une gestion efficace des données.

Pour notre part, nous regrettons que l'utilisation de serveurs distribués ne soit pas plus exploitée, par exemple, en stockant localement les tuples ou en permettant une redondance de serveurs. Nous avons l'impression que toute cette mécanique a été mise sur pied pour pouvoir gérer les deadlocks et le garbage collector qui demandent une énergie conséquente.

Nous estimons aussi que la documentation aurait pu être plus abondante en raison de la complexité de l'architecture. La gestion des deadlocks est inextricable : l'implémenteur jongle en effet avec les threads, les requêtes et les messages entre serveurs. Cette gestion est en plus répartie entre plusieurs modules (fichiers). On aurait aimé qu'il explique quelle stratégie il utilise. Nous sommes d'ailleurs interrogatif, comme l'auteur d'ailleurs le laisse entendre, sur le fait que sa stratégie ne provoque pas des comportements inattendus (bugs) ou n'arrive tout simplement pas à détecter tous les deadlocks.

Dans la distribution, nous pouvons trouver des fichiers qui proposent des **tests**. Mais ces fichiers sont curieusement dans le dossier examples. Face à notre étonnement, nous avons investigué et, dans la version téléchargeable sur `http://pylinda.googlecode.com/svn/trunk/pylinda-read-only`, nous avons trouvé un dossier de tests très complet.

Malgré les tests fournis, une amélioration ou une extension du code restent périlleuses. A notre sens, certaines parties du code devraient être toilettées et documentées, sinon repensées. Citons par exemple la gestion des deadlocks.

8.5.1 L'api

L'api est destinée aux programmeurs Python. Aucune autre information n'est donnée.

8.6 Pour conclure

Pylinda offre une implémentation bien structurée dans son ensemble, mais peu documentée. Pylinda offre un set de primitives suffisant pour expérimenter le langage de coordination Linda. Les performance sont raisonnables et permettent sans problème de réaliser de beaux programmes en Linda.

Chapitre 9

Analyse Pybrenda

9.1 Architecture logicielle

A la lecture des fichiers INSTALL et README, on comprend que l'architecture est de type RPC. Mais c'est en parcourant le code lui-même que l'on se rend mieux compte de la configuration que l'auteur utilise.

L'architecture part du principe que les processus utilisent une unité de stockage des données commune. Cette unité peut être un disque réseau mappé ou n'importe quel dispositif similaire[1]. Les processus eux-mêmes sont démarrés à partir de cette unité commune. On le voit, les contraintes sont très fortes et nous sommes loin d'une architecture réseau souple et ouverte.

Cette implémentation utilise le schéma master/worker. Cette liaison est très forte parce que les workers sont impérativement créés par le master. De plus, le master initialise et termine le serveur qui a en charge l'espace de tuples. On le voit, le couplage entre les différents composants est très fort.

Cette implémentation fonctionne sous deux modes :
- le mode local ;
- le mode distribué.

Le mode local utilise la technique du fork pour créer de nouveaux processus. Rappelons en effet que Python n'est pas un interpréteur multi-threadé et que donc le fork est quasi nécessaire si l'on souhaite exécuter du code en parallèle sur plusieurs processeurs.

Le mode distribué est lui implémenté par le recours à la commande rsh. Une liste des noeuds permet au master d'exécuter du code à distance (RPC). Cette exécution est toutefois soumise à quelques contraintes en raison des choix de l'implémentation. Citons les limites que l'auteur lui-même indique :

- la fonction passée à Eval() doit être du code ascii Python non compilé ;
- cette même fonction doit être définie au plus haut niveau du code. Il est donc impossible d'utiliser des classes ou des méthodes de classe ;
- l'utilisation d'objets de type fichier ouvert n'est pas possible.

Pour la persistance des données, tout se fait en mémoire. Par conséquent, comme le serveur est instancié par le programme principal, en cas de coupure, nous perdons toutes les données qui pourraient encore se trouver sur le serveur de tuples. Il y une sérialisation des données mais ceci est dicté par un choix d'implémentation et cette fonctionnalité n'est pas utilisée pour stocker les informations de manière durable.

[1] scsi, ethernet, smb, cifs, nfs

L'implémentation des différents composants est peu robuste. Par exemple, le client du serveur de tuples doit avoir un accès quasi instantané au serveur pour lire un tuple, ce qui est impossible sur un processeur à un seul coeur ou sur un processus distant.

La trop forte imbrication des différents composants est aussi un handicap. Par exemple, c'est le master qui crée le serveur de tuples et les workers. C'est une manière de faire qui n'est plus d'actualité puisqu'on essaye aujourd'hui de diminuer le couplage.

9.2 Le langage de coordination Linda

Seules les quatre primitives de base sont implémentées via la classe TupleClientCmds|Rsh[2]. Leur syntaxe est la suivante :

- out → instanceOfTupleClient.Out(arg1, ..,argN) ;
- in → instanceOfTupleClient.In(arg1, ..,argN) ;
- rd → instanceOfTupleClient.Rd(arg1, ..,argN) ;
- eval → instanceOfTupleClient.Eval(preTuple[3], function, arguments[4],postTuple=()[5]).

Les **tuples vivants** sont comme nous pouvons le constater bien présents. L'auteur explique de manière précise leur usage. Cette primitive accepte donc trois tuples et une chaîne de caractères pour l'identification de la fonction. Rappelons que le dernier paramètre (postTuple=()) est la manière de noter un paramètre par défaut en Python.

L'unification avec les **patrons** est faite sur le modèle du préfixe. Cela nous semble assez restrictif. Pour récupérer n'importe quel tuple, on utilise le tuple vide qui est préfixe de tout tuple.

9.3 L'espace de tuples

L'espace de tuples ne comporte aucune gestion. Ainsi, le fait que son implémentation ne soit pas indépendante du processus principal ne permet pas la communication entre plusieurs masters. Sa distribution sur plusieurs sites est, par design, impossible.

Son organisation est faite comme un fichier où chaque ligne représente un tuple qui est sérialisé.

La recherche dans ce fichier est donc de plus en plus consommatrice de temps au fur et à mesure que le fichier s'allonge.

9.4 La distribution de l'implémentation

9.4.1 L'installation

L'implémentation est livrée sous la forme d'un fichier tarball compressé. Après décompression, nous obtenons une arborescence de onze fichiers et de deux dossiers (un dossier pour les tests unitaires [6]

[2]Par abus de langage, nous appellerons TupleClient le TupleClientCmds|Rsh.

[3]C'est un tuple.

[4]C'est un tuple.

[5]C'est un tuple.

[6] *Tests unitaires* est pris dans son acception *senso lato*, c'est-à-dire comme tout dispositif qui permet de vérifier une non-régression du code ou un diagnostique au niveau du fonctionnement de ce code.

et un dossier pour les exemples).

```
|-- examples
|   `-- hello.py
`-- tests
    |-- test1.py
    |-- test2.py
    |-- test3.py
    |-- test4.py
    `-- testServerCds.py
|-- Brenda.py
|-- INSTALL
|-- LICENSE
|-- Makefile
|-- NEWS
|-- QuickStart
|-- README
|-- TODO
|-- einRpc.py
|-- myMath.py
|-- niftySocket.py
```

Le fichier **INSTALL** explique l'installation. On peut diviser ce fichier en quatre parties :
- la décompression des sources et leur installation ;
- la configuration des variables d'environnement ;
- l'accès aux noeuds de calcul via la commande rsh ;
- la vérification de l'installation.

La **décompression des sources** et leur installation dans les répertoires ad hoc est déjà problématique puisque les informations données sont trop floues. L'auteur explique par exemple que l'on doit copier les sources dans le dossier site-packages de son installation. Ce dossier dépend donc d'une installation à une autre et requiert des compétences d'administrateur système que tout le monde ne possède pas. Dans notre cas, avec une ubuntu 8.04.3, ce répertoire est situé ici : /usr/lib/python2.5/site-packages/.

La **configuration des variables** d'environnement permet quant à elle d'acccéder à l'implémentation. Notons que les informations fournies sont destinées à un public averti qui maîtrise les droits Unix et plus généralement la configuration d'un système Unix.

La **configuration des noeuds** de calcul se fait via un fichier que l'on édite. L'auteur propose une rapide vérification des noeuds via rsh.

L'auteur indique ensuite quelques pistes pour **vérifier son installation** qui n'est pas directement fonctionnelle actuellement comme nous le verrons au point suivant.

9.4.2 La prise en main

Cette implémentation n'est pas directement utilisable après son installation. En effet, quatre obstacles majeurs se présentent à l'utilisateur :

- selon les distributions, un module pour le langage Python est absent ;
- pour la définition des blocs, le code-source mélange les espaces et les tabulations et il empêche un comportement sain tant au niveau de l'édition du code que de son exécution. En effet, Idle (interface graphique de Python) refuse d'exécuter le code à cause de cette mixité ;
- le code-source comporte des erreurs ;
- il existe des présupposés architecturaux non explicites.

Le module manquant est **whrandom.py**. Ce script est un générateur aléatoire de nombres comme l'explique la documentation que l'on trouve à `http://pydoc.org/2.4.1/whrandom.html`. Ce fichier était

encore présent dans la distribution Python jusque dans la version 2.4.1 mais il a disparu, semble-t-il, dans les versions supérieures. Nous avons donc dû récupérer ce fichier.

Pour corriger la **définition des blocs**, nous avons été obligé de supprimer toutes les tabulations et de revoir la logique du code. Heureusement, comme nous le verrons dans la partie code, l'auteur indiquait toutes les fins de bloc par des commentaires.

L'**architecture logicielle** de type **RPC** utilise la commande rsh. Mais pour des problèmes de sécurité, comme le remarque d'ailleurs l'auteur, cette commande a été remplacée par ssh dans les distributions modernes. En effet, on ne peut plus accepter aujourd'hui que des processus soient déclenchés sans authentification.

Pour simuler le comportement d'une *authentification automatique*, comportement de rsh par ssh, nous pouvons utiliser une authentification par certificats. Nous avons choisi cette manière de faire avec un certificat qui est une clé asymétrique.

Cette clé asymétrique peut être créée avec ou sans mot de passe. La paire de clés créée, il suffit de distribuer la clé publique sur chaque machine distante *esclave*. C'est cette solution que nous avons utilisée pour tester l'implémentation.

Si l'on opte pour une clé avec mot de passe, il faut mettre en place un démon de type ssh-agent qui mémorise le mot de passe et le fournit à la demande.

La seconde problématique soulevée par l'**architecture du logicielle** - l'utilisation de disques réseaux mappés - peut être résolue grâce aux nombreux dispositifs disponibles tels que iscsi, nfs ou cifs. Rappelons que le script doit lui-même être exécuté à partir de ce disque mappé puisque l'implémentation part du principe que l'arborescence des fichiers doit être accessible et identique sur tous les noeuds de calcul.

Les **bugs du programme** sont de deux types :

- ceux liés à l'**archaïsme** de l'implémentation qui était prévue pour du code Python 1.5 alors que nous sommes à la version 2.5 [7]. Nous avons corrigé les appels aux fonctions sock.bind() et sock.connect() qui requièrent aujourd'hui un tuple en argument.[8] Nous n'avons pas touché aux autres archaïsmes ;
- les autres bugs sont liés à la conception même du programme. L'auteur gère en effet de manière très cavalière les liaisons entre les clients (qui sont ici des files sockets) et le serveur de tuples qui est démarré par le *programme principal*. Nous avons dû ajouter une temporisation au moyen de l'instruction time.sleep(3) avant la fermeture de la liaison par le serveur afin de permettre au client de récupérer l'information et ainsi clore son programme. Nous avons aussi modifié la fin des processus en ajoutant la commande os.kill.

La **documentation** est essentiellement composée par les trois fichiers **README**, **QuickStart** et **INSTALL** qui ne sont malheureusement pas suffisants pour utiliser cette implémentation. Une plongée au coeur même du code aidera les plus persévérants ou les plus téméraires.

[7] C'est la version en production qui est utilisée actuellement. Nous n'ignorons pas que la version 3.1 existe mais cette dernière n'est pas encore très répandue.

[8] Ancienne version pour bind : sock.bind(host, port).

9.5 Les sources

9.5.1 Le code

Le langage utilisé

L'**implémentation est faite en Python**. On remarque facilement que l'auteur est un programmeur C ou est fortement influencé par la syntaxe du langage C : alors que Python utilise l'indentation pour délimiter les blocs, l'auteur utilise une notation à la C pour signifier la fin d'un bloc. Voici un exemple qui illustre cette manière de faire :

```
def __serverLoop(self, listenSock):
    while self._serverUp:
        readyList = select.select([listenSock] + self._clientConns, [], [])
        for sock in readyList[0]:
            # sock is either listenSock or a client connection
            if (sock is listenSock):
                conn, addr = sock.accept()
                print ":ts got conn", conn, "from", addr
                self._clientConns.append(conn)
            else:
                #print ":ts got request from", sock
                self._dispatchRequest(sock)
                # end if
            # end for
        # end while
    # end def __serverLoop
```

Le coeur du programme est le fichier Brenda.py qui implémente les trois classes principales suivantes :

- TupleServer qui crée le serveur de tuples. Le serveur est instancié en même temps que démarre le script Python ;
- TupleClientsCmd qui crée la connexion au serveur de tuples. Cette classe est utilisée lorsque l'application tourne en mode local. C'est elle qui implémente les primitives de base de Linda, soit Out(), In(), Rd() et Eval(). Elle sert aussi de base à TupleClientRsh ;
- TupleClientRsh qui hérite de TupleClientsCsd et qui ne redéfinit que la méthode Eval() qui doit s'exécuter sur des noeuds distants et qui fait appel pour cela à la classe SimpleRpcRequest que l'on peut trouver dans le fichier **einRpc.py** et qui se charge de contrôler la syntaxe et de faire l'appel distant.

Nous trouvons aussi un autre fichier, **niftySocket**[9], qui a servi a résoudre un bug que l'auteur a rencontré avec la classe socket.

L'auteur fournit dans un répertoire un jeu de **cinq tests** que l'on peut ranger en trois catégories :

- un test qui vérifie le fonctionnement du serveur de l'espace de tuples ;
- deux tests pour le fonctionnement en local ;
- deux tests pour le fonctionnement en mode distribué.

La **documentation** proprement dite du **code** est suffisante et le code est très clair. Le souci de l'auteur de permettre la relecture et la compréhension de son code est manifeste. Cette clarté nous a permis de corriger par nous-mêmes les quelques bugs qui nous empêchaient d'utiliser l'implémentation.

[9]Nifty peut être traduit par débrouillard.

9.5.2 L'api

Une seule api en Python est disponible. Aucune indication n'est fournie pour construire ou pour porter le code vers d'autre langages.

9.6 Pour Conclure

Pybrenda est une implémentation simple mais elle nous semble peu adaptée à une utilisation réelle du langage de coordination de Linda.

Ses limites sont nombreuses. Citons entre autres :
- seules les primitives de base sont implémentées ;
- les requêtes sont préfixées sur les patrons ;
- l'architecture logicielle est vieillotte ;
- le couplage fort des composants rend notamment difficile le débogage ;
- la mise en oeuvre de l'implémentation nécessite des connaissan0ces profondes en architecture réseau et en administration système ;
- la quasi nécessité d'utiliser uniquement le modèle master/worker.

On le voit, cette implémentation est plutôt réservée au jeune hacker curieux de découverte des temps passés.

Chapitre 10

Analyse Linuxtuples

10.1 Architecture logicielle

Cette implémentation est une application client-serveur. A l'origine, elle acceptait jusqu'à 64 clients.

Le serveur utilise les sockets Linux et est multi-threadé. La politique des locks est simple : le premier arrivé est le premier servi. Pour garantir l'atomicité des opérations, chaque client possède un accès au tupleSpace de manière exclusive. Cette exclusivité est garantie au moyen du système de sémaphores. Le serveur garantit la cohérence du tupleSpace. Signalons que nous n'avons pas vérifié que les opérations sont bien atomiques mais.... c'est le contrat que le logiciel fournit.

Le serveur tient une liste des clients en cours. Cela lui permet de "libérer" les clients qui sont en attente lorsque le tupleSpace reçoit les données réclamées par les clients.

Chaque client peut avoir accès au TupleSpace simplement en s'y connectant. Il n'y a pas de session avec nom d'utilisateur ou mot de passe.

Toutes les communications entre le serveur et le client se font en clair.

D'un point de vue pratique, les tuples sont des listes doublement chaînées. La volonté ici n'est pas d'optimiser la recherche en utilisant un b-tree, un hash ou d'autres structures plus efficaces. Cela implique évidement que la recherche d'un tuple devient de plus en plus longue au fur et mesure que les tuples s'agrandissent.

Le tupleSpace est totalement en mémoire. Il n'y a donc pas de persistance des données.

L'implémentation du serveur est légèrement optimisée. En effet, les options
telles que TCP_NODELAY et SO_REUSEPORT apparaissent.

L'option TCP_NODELAY permet simplement de construire des trames plus importantes quand les données à transmettre sont de petite taille.

Quant à l'option SO_REUSEPORT, elle autorise les processus du serveur à utiliser la même adresse.

Le serveur ne peut pas être découvert : il n'existe pas de service d'annuaire.

Le serveur comprend la sémantique Linda qui est donc entièrement implémentée côté serveur.

En conclusion, Linuxtuples est une implémentation simple d'une architecture client-serveur.

10.2 Le langage de coordination Linda

Cette implémentation offre les primitives du langage de coordination Linda à l'exception de eval(). En effet, l'auteur signale que la solution adoptée par Gelernter lui paraît *inélégante*. Il propose plutôt d'utiliser ssh ou scp ou plus simplement NFS qui sont des commandes de base présentes dans toutes les installations de type Unix.

L'auteur a simplement changé le nom des primitives mais la sémantique est celle généralement admise.

Voici les primitives de base :

- out → get(template) ;
- in → put(template) ;
- rd → read(template).

Nous y trouvons aussi deux directives non bloquantes qui sont :

- inp → read_nonblocking(template) ;
- rdp → put_nonblocking(template).

Le copy et le copy-collect sont donc absents. En effet, un seul espace de tuples est géré par l'implémentation. Par contre, nous trouvons une directive :

- REPLACE (template, replacement). On peut imaginer l'utiliser dans la même optique que la commande copy.

Pour construire un patron, nous disposons de la directive None qui permet l'unification avec n'importe quel type de données.

Nous avons également accès à deux directives : dump() et get(). (Voir point suivant.)

10.3 L'espace de tuples

Pour la gestion de l'espace de tuples, l'auteur met trois commandes à disposition :

- dump() : qui retourne la liste des tuples présents sur le serveur. Cette directive accepte une liste de patrons ;
- count() : qui retourne le nombre de tuples dans l'espace de tuples. Cette directive accepte une liste de patrons ;
- log() : qui donne les informations sur l'état du serveur.

Ces trois commandes peuvent être données en argument à un script Python du nom de jobcontrol.py et produisent le même résultat. Ce script donne un choix intéressant en plus des commandes suivantes :

- empty : qui réinitialise l'espace de tuples ;
- stop : qui arrête tous les jobs ;
- test : qui vérifie si le tuple est actif ;
- start : qui permet de faire exécuter un script sur plusieurs slaves. Cette dernière option est très intéressante. En effet, la configuration se fait via le fichier SLAVES.

La réplication des serveurs n'est pas prévue.

10.4 La distribution de l'implémentation

10.4.1 L'installation

L'implémentation est livrée sous la forme d'un fichier tarball compressé. Après décompression, on obtient une arborescence plate de seize fichiers que voici :

```
|-- linuxtuples-1.03
|   |-- ChangeLog
|   |-- Doxyfile
|   |-- Makefile
|   |-- README
|   |-- SLAVES
|   |-- endian_test.c
|   |-- fft.c
|   |-- jobcontrol.py
|   |-- package.html
|   |-- pingpong.py
|   |-- py_linuxtuples.c
|   |-- testCount.py
|   |-- tuple.c
|   |-- tuple.h
|   |-- tuple_client.c
|   `-- tuple_server.c
```

Un rapide coup d'oeil au fichier README nous informe qu'il faut compiler les sources pour avoir un fichier exécutable pour le serveur et un autre fichier pour la librairie dynamique utilisable sous Python.

Même si cette manière de faire est courante dans le monde Open Source et plus particulièrement Linux, cela rebute l'utilisateur final non informaticien qui ne sait pas qu'il faut installer les outils nécessaires pour une compilation. Mais l'avantage de cette solution est que l'on peut améliorer le code si on le souhaite.

La **prise en main** est rapide. Les exemples donnés dans le fichier README sont compréhensibles. L'auteur donne en plus quelques conseils pour programmer en parallèle.

La **documentation** est suffisante mais elle ne livre que peu d'exemples. On sent que l'auteur s'adresse à un public de programmeurs. On peut utiliser du C pour la programmation de son client. L'api Python est simplement là pour ceux qui ont envie de tester rapidement l'implémentation.

Signalons que l'auteur offre la possibilité de tester son implémentation en calculant une fft sur des nombres produits au hasard par /dev/random. C'est une très mauvaise idée car les deux processus ne sont pas du tout équilibrés. La production de nombres aléatoires est très rapide et la mémoire est rapidement encombrée. Quant à l'implémentation de la fft, elle n'est pas du tout optimisée. La fft ne sait donc pas suivre le rythme et la machine s'effondre par manque de mémoire.

10.5 Les sources

10.5.1 Le code

Les sources sont écrites en C. La documentation du code est claire et concise. Nous pouvons apprendre beaucoup en parcourant le code. La mise en page du code est plaisante à la lecture. On voit que l'auteur propose un listing lisible au commun des mortels.

Nous trouvons un test unitaire testCount.py qui se réduit à sa plus simple expression. Deux directives sont testées, soit le get() et le put(). Les autres directives ne sont pas testées.

10.5.2 L'api

Deux api sont disponibles : la première en C et la seconde en Python. Il n'existe pas de documentation explicite pour construire d'autres apis.

L'extension du programme est toutefois réservée à des programmeurs qui connaissent à la fois le C et le Python et qui savent comment on interface les deux.

10.6 Pour conclure

Cette implémentation est simple et offre déjà quelques possibilités mais on sent que l'on peut être rapidement limité :

- par la construction du patron qui ne permet pas de distinguer le type de variables que l'on veut unifier ;
- par le manque de certaines directives comme le copy-collect ;
- par l'absence de sérialisation des données ;
- par l'absence de persistance ;
- par l'existence d'un seul espace de tuples.

Chapitre 11

Analyse NetwokSpaces

11.1 Architecture logicielle

Cette implémentation du modèle Linda est arrivée à maturité. Cette maturité se remarque à la clarté de sa documentation mais surtout à son découpage, à son architecture logicielle et aux choix des composants qui l'implémentent.

Networkspaces utilise le modèle client-serveur comme base de son architecture. Sa structure lui permet de faire tourner plusieurs serveurs en parallèle sur la même machine.

La construction de son serveur utilise Twisted. Twisted est un framework orienté événements qui est écrit en Python. Il utilise le modèle asynchrone. Lorsqu'un client fait une demande qui est potentiellement bloquante, le serveur envoit immédiatement un réponse. Le serveur utilisera ensuite la technique du *callback* pour fournir le service demandé par le client.

Twisted sépare la couche transport et la couche protocole. C'est un outil orienté production qui permet de construire rapidement des clients et des serveurs utilisant une vaste palette de protocoles. Il donne en outre la possibilité de créer rapidement tout type de protocole.

Comme sa description plus avant dépasserait le cadre de notre mémoire, nous invitons le lecteur curieux de cette technologie à se rendre sur le site `http://twistedmatrix.com/trac/`.

Le modèle proposé par Twisted est totalement adapté à l'implémentation du modèle Linda. En effet, les primitives bloquantes vont ainsi être optimisées. D'un côté le serveur peut traiter d'autres requêtes mais surtout il peut fournir de manière optimisée le client si l'information qu'il a demandée arrive dans l'espace de tuples.

Le découpage de l'implémentation peut être faite en quatre modules :

- le serveur d'espace de tuples ;
- les apis clients ;
- une interface graphique de l'activité du serveur ;
- un module de parallélisation du code.

Comme nous l'avons vu, le **serveur** est un serveur asynchrone utilisant Twisted. Il est totalement autonome des autres modules.

L'**api client** est développée dans trois langages de script qui sont Python, R et Matlab.

Une **interface Web** permet de visualiser l'activité du serveur et de faire de la maintenance.

Cette implémentation propose un module assez évolué permettant d'exécuter du code sur d'autres noeuds. Elle reprend l'idée du RPC mais en le modernisant grâce à l'utilisation :

- d'itérateurs qui permettent d'exécuter soit une seule fois une fonction sur plusieurs noeuds soit plusieurs fois la même fonction mais avec des paramètres différents ;
- du protocole ssh qui permet une meilleure authentification.

A l'origine, l'architecture n'était pas résistante aux pannes. Mais actuellement, il est facile lorsqu'une application est correctement architecturée de lui permettre de résister aux pannes.

La persistance des données est partiellement implémentée. En effet, si les données peuvent être conservées par le serveur pendant toute la durée de vie du serveur et non du processus qui les a créées, nous n'avons pas découvert comment enregistrer les données à la fermeture du serveur. Cette fonctionnalité peut être implémentée de manière rapide parce que les données sont sérialisées.

11.2 Le langage de coordination Linda

Même si nous avons ici un langage de coordination basé sur un ensemble de primitives et sur un espace commun d'échanges, Newtworkspace diffère des autres implémentations quant à la notion de tuple. Ici ce sont bien des variables qui sont stockées dans l'espace commun. L'implémentation peut même garantir l'unicité de chaque variable.

Nous sommes donc en présence d'un modèle qui se rapproche plus du modèle clé-valeur [1] pour la gestion des données. La suite du chapitre doit être lue en conservant cette idée à l'esprit.

Les **primitives de base** sont évidemment implémentées de manière directe (out, in, rd) et de manière indirecte (eval).

Les tuples sont accédés sous différents modes : fifo ou lifo par exemple. Cela lève l'ambiguïté de certaines primitives comme par exemple rd qui devient sémantiquement proche de *copier* plutôt que d'*accéder*. Les commandes copy et copy-collect du modèle initial sont aussi de ce fait obsolètes.

- out → store(X,V) : associe la valeur de V à la variable X ;
- in → Fetch(X) : copie et supprime la valeur (requête bloquante) ;
- rd → Find(X) : copie la valeur (requête bloquante) ;
- eval → utilise le mécanisme mis en place par la classe Sleigh.

11.2.1 Les tuples vivants

La classe Sleigh implémente le mécanisme des tuples vivants. Cette classe permet d'exécuter du code sur des noeuds distants. Elle offre de nombreuses possibilités et la granularité de ses fonctions lui permet par exemple de définir un environnement de travail par noeuds.

11.2.2 Les primitives supplémentaires

- inp → FetchTry(X) : copie et supprime la valeur (requête non bloquante) ;
- rdp → Find(X) : copie la valeur (requête non bloquante).

[1] Keys-Value.

11.2.3 Le patron

La notion de tuple a laissé la place à la notion de clé-valeur. Pour accéder aux valeurs, il suffit donc d'indiquer la variable[2] ainsi que l'espace de tuples[3].

Cette évolution est, sinon opportune, dans l'ère du temps. Les plus éminants chercheurs en Informatique essaient aujourd'hui d'explorer ce paradigme clé-valeur. Il apporte plus de souplesse que le modèle base de données et offre grâce à la sérialisation une très grande souplesse d'échange d'informations.

Nous avons l'impression que, cette nouvelle manière de voir les tuples correspond à l'état d'esprit que était celui des créateurs de Linda : apporter un nouveau paradigme pragmatique. Nous ne prendrons pas position dans ce débat que les puristes ne manqueront pas d'alimenter.

11.3 L'espace de tuples

L'espace de tuples peut être géré via une interface Web. Le programmeur dispose en outre de plusieurs directives pour le gérer :

- **deleteVar** : supprime une référence à un tuple ;
- **deleteWs** : supprime un espace de tuples ;
- **ListWss** : liste les espaces de tuples disponibles.

Il n'est pas possible de créer un **cloud computing** en fédérant plusieurs serveurs.

11.4 La distribution de l'implémentation

11.4.1 L'installation

L'installation se fait en deux parties :
- l'installation du serveur ;
- l'installation de l'environnement client.

L'installation du serveur est notamment expliquée dans le fichier INSTALL du paquet serveur.

L'installation requiert des prérequis qui sont, outre Python :

- twisted-web [4] ;
- twisted [5] ;
- zope interface [6].

Après installation des prérequis, il faut décompacter les fichiers, construire l'installation Python (qui nécessite le paquetpython-dev[7]) et enfin installer l'ensemble.

Signalons qu'il est possible d'effectuer une installation pour un utilisateur en particulier ou pour tous les utilisateurs.

---

[2] Clé.
[3] Espace de clés-valeurs.
[4] apt-get install python-twisted-web installe le paquet sur une Debian like.
[5] apt-get install python-twisted installe le paquet sur une Debian like.
[6] apt-get install python-zopeinterface installe le paquet sur une Debian like.
[7] apt-get install python-dev installe le paquet sur une debian like.

Nous regrettons que ce fichier ne donne aucune indication permettant de vérifier que l'installation s'est bien déroulée. Pour ce faire il faut lire le fichier README qui explique comment démarrer le serveur et comment accéder à son interface Web.

L'interface Web a besoin de **pybabelfish** pour permettre à l'utilisateur de lire le contenu des variables. Ce programme se lance simplement en tapant la commande pybabelfish sur la ligne de commande avec les droits root.

Dans le fichier, nous trouvons plusieurs méthodes pour démarrer le serveur. Il y est fait mention d'un script nws qui permet facilement de démarrer et d'arrêter le serveur. Mais l'auteur oublie d'indiquer où se trouve ce fichier. Ce dernier est en fait situé dans le dossier misc de l'archive décompressée.

Les deux fichiers **README** et **INSTALL** permettent donc une installation sans problème. Les indications sont claires et l'on sent que des efforts ont été fournis pour donner dès le départ une impression de professionalisme.

L'installation de l'environnement client est, comme on pourrait s'en douter, dans la même philosophie simple et rapide. Ici il n'y pas de prérequis et un simple Python setup.py install après décompression du fichier rend le client opérationnel sur une machine. Si l'on souhaite travailler avec des noeuds, une configuration supplémentaire est nécessaire.

La **prise en main** de ce logiciel est facilitée par une documentation de 64 pages dont la qualité est surprenante. Pour s'en convaincre, j'invite le lecteur à lire la dernière page qui décrit la configuration d'un login ssh avec authentification asymétrique sans mot de passe.

Les exemples et les explications qui sont fournis par la documentation permettent un démarrage rapide. Ajoutons que cette implémentation est très complète.

11.5 Les sources

Les sources pour la **partie serveur** sont écrites en Python. Par manque de temps, nous n'avons pas pu analyser en profondeur le code. Le serveur utilise, comme nous l'avons signalé dans l'analyse, le framework Twisted.

Ce dernier est framework de développement. L'utilisation d'un framework ne permet généralement pas à un non-initié une compréhension fine du code et ceci à notre avis pour les trois raisons suivantes :

- les frameworks sont souvent des mondes clos ;
- les framework cachent la complexité du développement en utilisant des paterns ;
- des conventions implicites sont mises en avant par certains frameworks.

La **fermeture des framework** sur eux-mêmes donne l'impression que la documentation du code est peu explicite. Ce *manque* doit être relativisé. En effet, selon que l'on se place côté développeur maîtrisant son framework ou côté analyste extérieur sans connaissance du framework, la qualité de la documentation est sujette à appréciation : on voit mal le développeur aguerri au framework documenter ce qu'il considère comme l'abc de son outil.

Les **patrons de conception**[8] sont souvent la base même d'un framework. En effet, pour cacher la complexité du développement, les frameworks encapsulent dans des couches d'abstraction des concepts dont la mécanique est souvent peu ou pas connue. C'est un des objectifs d'un framework. Cela ne facilite évidemment pas la compréhension du code pour une personne extérieure.[9] Pour Twis-

[8]Design pattern.

[9]Même si le pattern est connu, citons par exemple le pattern Modèle-Vue-Contrôleur.

ted, le patron de conception utilisé est la fabrique[10]. Ce patron de conception permet de créer de manière dynamique des sous-classes en passant par des intermédiaires. Twisted utilise également la notion d'interface qui est bien connue en Java mais qui est moins courante en Python.

Les conventions implicites sont mises en avant dans certains frameworks. Cette course à l'implicite est d'autant plus forte que Python, mais aussi Ruby, utilisent déjà l'usage de l'implicite dans la déclaration de leurs variables ou dans l'usage de leurs classes. Python, par exemple, utilise le **type ducking** [11].

Les frameworks de type Twisted étendent la notion de l'implicite à la construction d'une application entière. En effet, il suffit d'assembler les différentes briques du framework pour construire rapidement soit un protocole, soit une interface Web, soit un serveur. Cela n'est possible que parce que l'on connaît les conventions d'utilisation de ces briques.

Pour illustrer notre propos, Twisted renvoit en cas d'action bloquante un objet *Deferred* auxquel il est possible d'attacher des actions. Pour ce faire, cet objet possède une méthode de classe *addCallback* qui prend comme premier argument la fonction et donne la possibilité de définir des arguments pour la fonction. On le voit, la mécanique du framework est complexe.

En conclusion, si le framework accélère le développement de l'application et donne une liberté aux développeurs, cela ne permet pas à une personne extérieure de comprendre le code à la lecture.

On le comprend, l'analyse du code de Networkspaces passe d'abord par l'intégration des concepts et des conventions du framework Twisted. Ne disposant pas du temps nécessaire pour nous approprier cet outil, nous nous limiterons à présenter l'organisation générale du code.

Le code est découpé de manière fonctionnelle :

- protocol.py implémente un nouveau protocole de communication ;
- server.py implémente le serveur d'espace de tuples ;
- web.py implémente le serveur Web qui permet de visualiser l'activité de l'espace de tuples.

Quant à l'api client qui permet d'interagir avec le serveur, la documentation du code est exceptionnelle. Les explications permettent de comprendre le code qui est fourni.

Nous trouvons des tests dans le module clients.

11.5.1 L'api

Outre l'api client en Python, il existe une api pour Matlab et pour R. Nous n'avons pas eu le temps d'explorer ces deux apis.

11.6 Pour conclure

En conclusion, l'installation et la prise en main sont simples en raison de la complexité qui se cache derrière les outils.

C'est une implémentation très complète qui se distingue de toutes les autres par son offre importante de fonctionnalités, par son architecture et surtout par sa vision de la notion d'espace de tuples qui est devenue un espace de variable-valeur.

[10] Factory.

[11] Expression que nous pouvons traduire par : si une classe fait coincoin, c'est que c'est une classe Canard.

Troisième partie

Implémentation

Introduction

Cette partie présente notre implémentation d'un langage de coordination inspiré par le modèle Linda. Nous l'avons implémenté comme vous l'aurez compris en Python. Ceci pour toutes les raisons que nous avons évoquées jusqu'ici.

L'accent a été mis sur la démarche de construction et l'analyse des possibles. Notre objectif n'est pas, dans un premier temps néanmoins, de concurrencer une autre implémentation mais bien de montrer les choix que nous avons faits et surtout de les justifier. Nous ne prétendons pas que nos choix sont meilleurs mais simplement qu'ils sont nôtres et que nous pouvons dès lors les défendre et les mettre en valeur.

Au-delà de l'aspect académique, puisqu'il faut voir cette implémentation dans le cadre d'un mémoire, nous trouvons intéressant de plonger *nos mains dans le cambouis* et de participer ainsi à une expérience que nous a enrichi.

Les différentes thématiques qui seront abordées dans cette partie sont celles de la gestion d'un projet : définition des besoins, élaboration d'un cahier des charges, choix des designs, application d'outils méthodologiques, découpage du projet, production d'un livrable et séquencement des activités.

Toutefois, si certaines thématiques sont explicitement abordées comme par exemple les choix méthodologiques ou les besoins, d'autres seront tout simplement ignorées. Dans cette catégorie nous trouvons la planification, la gestion de l'équipe ou bien la constitution des différents groupes de contrôle que l'on rencontre dans tout projet.

Notre objectif est de faire simple tout en gardant à l'esprit une certaine puissance de l'implémentation. Notre choix s'est donc porté vers une gestion de projet orientée agile. Précisons que nous n'avons pas choisi cette méthode parce qu'elle est *à la mode* mais bien parce qu'elle semblait s'imposer : nous ne voulions pas figer par une définition trop précoce et peut-être trop rigide les besoins auxquels devait répondre notre implémentation.

Chapitre 12

Cahier des charges

12.1 Introduction

Ce cahier des charges doit se lire comme l'état d'esprit qui nous a guidé pendant notre implémentation. Nous insistons ici à nouveau sur le fait que notre souci de ne pas figer trop rapidement notre analyse des besoins nous a obligé à donner des pistes plutôt que des exigences non négociables.

Définir soi-même un cahier des charges est périlleux. Nous sommes conscient que cette approche est quelque peu schizophrénique puisque l'implémenteur est aussi le client. Mais nous avons voulu au maximum corriger ce biais en nous basant sur l'expérience que nous avons acquise lors de l'analyse de l'existant.

La base de ce cahier des charges est donc l'analyse des implémentations mais complété par notre expérience de tous les jours.

Voici donc les lignes directrices qui seront le cahier des charges ainsi que sa justification :

- développement en utilisant des outils open source ;
- construction autour d'une architecture logicielle simple ;
- découpage modulaire du projet ;
- utilisation de tests unitaires ;
- respect des conventions d'écriture ;
- documentation du code ;
- compatibilité syntaxique avec Pylinda.

Remarquons le caractère techniques des exigences qui s'explique ici par le fait que nous developpons une API et un langage de coordination.

12.2 Ouverture au monde de l'open source

L'**ouverture au monde de l'open source** était pour nous une exigence importante. Lors de l'analyse de l'existant, nous avons pu apprécier l'accès au code d'autrui. Nous voulions aussi permettre à tout un chacun la même démarche de compréhension et d'analyse.

Il est également primordial de pouvoir utiliser librement certaines briques logicielles dans un projet pour ne pas à chaque fois réinventer la roue. Utiliser des outils libres de distribution et de modification était donc un gage d'investissement durable.

Nous voulions aussi permettre à n'importe qui de modifier le code sans faire d'investissement financier pour les outils.

12.3 Architecture logicielle simple

La **construction autour d'une architecture logicielle simple** de l'implémentation doit lui permettre de mieux être comprise par l'utilisateur. Durant notre analyse, nous avons remarqué que certains paradigmes étaient une entrave à l'utilisation. Vouloir par exemple une organisation identique pour le système de fichier est à notre sens une exigence qui ne peut exister aujourd'hui. L'architecture logicielle devra permettre à un agent de rejoindre simplement le système et ainsi participer avec un minimum de configuration.

12.4 Modularité du projet

Le **découpage modulaire du projet** doit permettre d'isoler le code qui pose problème, de favoriser une plus grande maintenance mais surtout de réduire le couplage entre certaines parties de l'implémentation. Nous pourrions à un moment ou à un autre vouloir implémenter le serveur d'espace de tuples avec une autre technologie. Le découpage en modules peut aussi permettre une meilleur organisation et une surveillance plus fine de l'état d'avancement de chaque brique. Le découpage modulaire permet donc une meilleure autonomie dans les équipes de développement.

12.5 Tests unitaires

L'**utilisation de tests unitaires** est un gage de qualité. Il permet surtout une sérénité tant pour le développeur que pour l'utilisateur. C'est un outil qui à notre sens est indispensable dans la construction de logiciels. La vérification de code est de toute manière une nécessité. Cette nécessité est ici d'autant plus marquée que l'utilisation d'un langage de coordination rend difficile le débogage d'application du fait qu'elle s'exécute en parallèle. Il est dès lors primordial de s'assurer que la moindre modification dans l'implémentation ne va pas produire des disfonctionnements. Les tests unitaires permettent aussi dans une certaine mesure de garantir une stabilité de fonctionnement.

12.6 Les conventions d'écriture

Les conventions d'écriture permettent une lecture facile aux personnes extérieures. Elles permettent donc une analyse ainsi qu'une compréhension plus rapide. Les conventions d'écriture permettent aussi une utilisation accélérée du code. Le code doit pouvoir être étendu facilement.

L'adoption des conventions d'écriture est donc un critère de qualité.

12.7 Documentation du code

La **documentation du code** si elle semble aller de soi, reste encore le parent pauvre des projets informatiques de petite ou moyenne taille. Nous avons donc voulu apporter un soin particulier à la documentation.

12.8 Compatibilité syntaxique avec Pylinda

La nouvelle implémentation doit assurer une compatibilité avec la version 0.6 de Pylinda. Cela est important pour pouvoir réutiliser le code qui a été écrit précédemment. Mais une compatibilité parfaite n'est pas exigée.

Chapitre 13

Choix d'implémentation

Introduction

Les choix d'implémentation font référence en partie aux exigences que nous avons décrites précédemment.

13.1 Architecture Logicielle

Pour respecter la contrainte du cahier des charges relatif à l'architecture (cfr 12.3 page 80), nous avons choisi le paradigme client-serveur et une séparation nette des différents composants.

13.1.1 Client-serveur

Le paradigme client-serveur nous semblait le plus approprié. C'est une architecture qui est arrivée aujourd'hui à maturité et qui est surtout très connue.

Nous voulions éviter par exemple la complexité d'une exécution distante via rsh ou ssh.

L'architecture client-serveur offre l'avantage de pouvoir facilement fonctionner sur des réseaux étendus privés ou publics.

La liaison la plus forte reste le protocole utilisé par le client et le serveur pour communiquer. Actuellement, des outils existent pour construire un protocole sans devoir construire la couche transport.

L'utilisation d'un protocole de communication est aussi un avantage, notamment si l'on souhaite loguer finement l'activité d'un serveur. Cela permet également de pouvoir rejouer certaines sessions.

Nonobstant le protocol, le couplage est relativement lâche entre le client et le serveur. En effet, nous pouvons à peu de frais passer d'un protocole de type ftp à un protocole REST [1], qui priviliégie le protocole HTPP.

Cette architecture permet aussi d'utiliser d'autres paradigmes sans modifications importantes :

- on peut voir les clients comme des agents et le serveur comme un lieu d'échange d'informations ;
- on peut utiliser le modèle master-worker (les clients) et le serveur comme lieu de coordination.

[1] Representational State Transfer.

Quant aux différents mécanismes de sécurité, ils sont facilement implémentables, que ce soit le tunneling ou simplement le cryptage des messages échangés entre le client ou le serveur.

Nous avons donc opté pour ce paradigme car il nous paraissait moderne, simple et surtout flexible.

13.1.2 Spécialisation des composants

Nous avons opté pour une séparation forte et une spécialisation des composants. Nous ne voulions pas mélanger l'api client et le code du serveur comme dans certaines implémentations. Nous voulions que chaque composant soit autonome.

Notre code sépare donc le serveur, l'api client lié au langage de coordination et les modules utilitaires.

Cette séparation forte des composants nous permet aussi de réutiliser des composants externes. Pour exécuter du code distant, nous pouvons par exemple utiliser la bibliothèque parallèle ssh.

Cette volonté de spécialisation des composants rejoint aussi la contrainte de modularité (cfr 12.4 page 80).

13.2 Découpage du projet

Le découpage du projet (contrainte 12.4 page 80) est le suivant :

- Redis : le serveur d'espace de tuples. Cette brique est un projet Open Source qui implémente un serveur de clé-valeur très performant. Nous l'avons choisi pour ses qualités intrinsèques et parce qu'il respectait notre contrainte d'utilisation de projet open source (cfr contrainte 12.2 page 79) ;
- RedLinda : l'API client pour les primitives de base à l'exception de eval. Nous n'avons pas intégré eval à notre API parce que le cahier des charges (cfr contrainte 12.8 page 81) ne l'exigeait pas mais surtout parce que nous voulions tester le même code pour les deux environnements. En implémentant la primitive eval de manière séparée, nous pouvions l'utiliser avec Pylinda et ainsi employer le même code pour tester les deux implémentations ;
- EvalRedLinda : la classe qui implémente les tuples vivants (eval).

13.2.1 Redis : le serveur

Redis est un projet qui implémente une base de données sur le modèle clé-valeur. Ce programme est écrit en ANSI-C pour les sytèmes comptables Posix.

Il est sous licence BSD.

On peut trouver le projet à l'adresse `http://code.google.com/p/redis/`. Ce projet est à la veille de proposer sa release candidate 1.0. Comme on peut le remarquer, c'est un projet jeune.

Nous l'avons retenu pour les raisons suivantes :

- ses performances sont impressionnantes. En effet, sur une machine actuelle, il peut répondre à plus 80.000 requêtes par seconde ;
- il offre nativement plus de 9 API pour les langages tels que Java, Ruby, Erlang, PHP et Python ;
- son code est clair, concis et documenté ;

- il supporte la réplication ;
- il offre une persistance des données ;
- il gère les chaînes de caractères, les listes et les ensembles ;
- il permet certaines opérations atomiques ;
- il fournit une documentation claire.

Ses nombreuses qualités nous permettent donc de disposer d'une base saine pour notre serveur de tuples. Et en choisissant Redis nous avions en plus :

- une persistance des données ;
- une distribution répartie sur plusieurs serveurs ;
- une réplication du serveur.

Une autre qualité de Redis est le faible poinds de son exécutable (245 Kb) qui fait que nous pouvons l'utiliser sur une **application embarquée**.

La **séparation des modules de l'implémentation de Linda** est aussi requise. Rappelons qu'un module[2] en Python est un regroupement de code sous forme de fichier.

13.2.2 Redlinda : API client

Le coeur de notre implémentation est le paquet Redlinda.

Pour construire notre implémentation, nous allons nous appuyer sur l'API Python de Redis.

C'est donc tout naturellement que notre classe TupleSpace, qui est la base de notre implémentation, va hériter de l'API Python de Redis. Remarquons toutefois que nous mettons à disposition toute l'API Redis. Même si cela paraît être un avantage pour l'utilisateur, il aurait été préférable de cacher l'API Redis, mais Python ne connaît ni les classes ni les méthodes privées.

TupleSpace est une classe qui opte pour les nouvelles spcécifications des classes en Python. Les classes *nouveau style en Python* héritent de la classe Objet. Il est important de signaler que ce choix implique que le code soit compatible avec les versions 2.2 et supérieures de Python. Ce désavantage tout relatif nous offre de meilleures possibilités pour la sérialisation des objets utilisée dans notre projet.

La classe TupleSpace permet donc d'instancier un espace de tuples et d'y appliquer les primitives du langage de coordination Linda.

Cette classe se veut compatible, comme l'exige le cahier des charges, avec la même classe implémentée en Pylinda [3] (voir 12.8 page 81).

Pour garder cette compatibilité, une première solution aurait été de surcharger la classe de Pylinda puisque Python est un langage objet qui permet l'héritage simple, le polymorphisme et la surcharge. Nous avons décidé de ne pas surcharger cette classe, ce qui nous aurait obligé à conserver les fichiers en permanence.

Une autre possibilité aurait été de créer une classe abstraite (interface en Java). Cela aurait permis de garantir, si l'on respecte le contrat, une interopérabilité entre les deux implémentations. Mais d'une part, cette pratique n'était pas très pythonesque et, d'autre part, il aurait fallu que le concepteur de Pylinda ait un intérêt à respecter un contrat que nous extrapolons sur base de son implémentation.

La classe TupleSpace implémentera donc les primitives Linda en les traduisant pour le serveur

[2] Séquences d'opérations directes, fonctions ou classes

[3] Dans l'implémentation Pylinda, cette classe se trouve dans le module kernel.py. Nous avons donc donné le même nom à notre module.

Redis que nous avons choisi comme container d'espace de tuples.

13.2.3 Evalredlinda : tuple vivant

Ce module abrite la classe EvalRedLInda qui ajoute la possibilité d'utiliser les tuples vivants. Nous avons choisi d'utiliser la richesse de Pyton qui propose à l'origine deux stratégies pour exécuter du code Python à l'intérieur d'un programme Python : l'utilisation d'une instruction (exec et execfile) ou l'utilisation d'une fonction (eval).

13.3 Méthodologie

Introduction

Dans cette sous-section, nous vous présentons différentes techniques que nous avons expérimentées au niveau méthodologique. Elles appartiennent toutes à la mouvance de la gestion des projets qui utilise des méthodes agiles.

Voici donc les techniques que nous allons vous présenter :

- développement guidé par les tests ;
- développement guidé par la documentation ;
- respect des conventions de codage.

13.3.1 Les méthodes agiles

Cette méthodologie a été choisie parce que nous ne voulions pas fixer trop rapidement les contraintes de développement.

Nous sommes conscient que sous le label *Agile* nous trouvons toute une famille de méthode. Mais nous sommes aussi conscient que les techniques ne sont pas un état d'esprit.

Notre position est plus proche du livre de Tarek Ziadé [4] [Ziadé, 2007] que celui de Véronique Messager Rota[5][Rota, 2009]. En effet, celle-ci envisage plus les relations entre les différents protagonistes d'un projet et de sa planification. Notre situation d'isolé nous conduit plutôt à profiter des conseils concrets et techniques que les méthodes agiles ont mis à l'honneur. Comme par exemple :

- une conception simple ;
- le remaniement fréquent du code pour rester aussi simple et clair que possible ;
- les tests unitaires (ou tests de non régression) ;
- les tests de recettes (ou tests de conformités aux exigences).

Notre objectif était clairement d'expérimenter des outils, de les vivre, de les confronter à notre quotidien et de nous former une opinion.

Comme nous voulions expérimenter le maximum de techniques, nous ne pouvions les appliquer toutes en même temps et à l'ensemble de notre démarche. C'est pourquoi nous avons cherché à mettre en adéquation la technique par rapport à la phase de développement. Nous avons par exemple utilisé

[4]Python : petit guide à l'usage du développeur agile.
[5]Gestion de projet : vers les méthodes agiles.

Pylint qui vérifie les conventions de codage sur deux programmes écrits dans le cadre de notre exploration de traitement du son. Par contre, les tests ont été au centre lorsque nous avons développé notre API client. Signalons que certaines techniques comme la génération automatique de documentation a été testée[6] mais que nous ne l'abordons pas dans le mémoire.

13.3.2 Développement guidé par les tests

Pour le langage Python, il existe un grand nombre d'outils de test de qualité destinés à simplifier la vie du développeur. Pour répondre à la contrainte 12.5 page 80, nous avions le choix soit d'utiliser les outils offerts par la distribution Python elle-même, soit d'utiliser des outils externes. Nous avons choisi d'utiliser un outil externe : *Pytest.* (Cfr 13.4.2 page 88).

Nous avons donc utilisé les tests unitaires pour notre module principal, soit kernel.py qui implémente notre classe TupleSpace.

Pour ce faire, nous avons élaboré deux séries de tests :

- une première série sur les primitives du langage qui se trouve dans le fichier test_primitive.py [7] ;
- une seconde série pour les erreurs[8] que nous trouvons dans le fichier *test_erreur.py*[9].

Les primitives du langage

Pour chaque primitive, nous avons essayé d'envisager un maximum de cas d'utilisation. Voisi un exemple pour la primitive in :

- verifier que l'on consomme bien un tuple défini par un patron ;
- verifier que l'on récupère un tuple qui existe en définissant tous les élements ;
- verifier que l'on récupère un tuple qui existe en utilisant un patron.

Le lecteur avisé aura remarqué que nous ne testons pas toujours toutes les combinaisons. La consommation n'est testée qu'une seule fois alors que nous aurions pu la tester avec une définition précise du tuple que l'on souhaite tester ou avec un patron.

Les classes d'erreur

Notre implémentation permet de paramétrer finement le comportement de l'API au niveau des délais de réponses. Nous voulions dès lors avoir la ganularité la plus fine possible pour les tests de disfonctionnement de notre API.

Pour ce faire, nous avons créé pour chaque paramètre ou groupe de paramètres une classe d'erreur qui hérite de la classe Execptions de Python. Toutes ces classes sont regroupées dans le module *kernel.py.*

Les tests simulent simplement les situations d'erreur et vérifie si les exceptions correspondantes sont bien déclenchées.

---

[6]Utilisation de l'outil Sphinx `http://sphinx.pocoo.org/` et des conventions d'écriture docstring reStructuredText http ://docutils.sourceforge.net/rst.html.

[7]Dans le dossier tests.

[8]Pour chaque type d'erreur nous avons donc créé une classe d'Erreur. Par exemple, si le temps de réponse du serveur est dépassé, cette erreur est gérée par unique classe d'erreur.

[9]Dans le dossier tests.

13.3.3 Développement guidé par la documentation

En utilisant le module evalredlinda qui implémente la classe EvalRedLinda, nous avons rencontré en même temps les exigences de test et de documentation (cfr point 12.7 page 81 et point 12.5 page 80).

Pour vérifier le fonctionnement de cette classe, nous avons choisi d'utiliser le développement guidé par la documentation. Cela nous paraissait adapté dans ce cas-ci. En effet, cette classe a peu de dépendance. Nous voulions aussi expliquer les mécanismes que nous utilisions. Cela nous obligeait à écrire de la documentation et à illustrer notre propos par des exemples concrets d'utilisation. Les doctest que propose Python étaient donc totalement adaptés dans ce cas-ci. Les doctest sont en fait un mécanisme qui permet de vérifier la documentation que l'on écrit. Si on copie le code d'une session à l'intérieur d'une explication, les doctest vont considirer que chaque ligne qui contient une réponse de l'interpréteur à la commande est un assert. Par exemple si nous écrivons ceci :

```
>>> a = 10
>>> print a
10
```

La sortie *10* est équivalente à *assert a==10.*

Au niveau pratique, nous faisons remarquer que les facilités offertes aux programmeurs sont loin d'être faciles à utiliser. Par exemple, dans un bout de log, on aimerait au moins connaître la ligne où l'erreur s'est produite. Cela oblige toutefois à écrire les tests un par un et à les vérifier.

Malheureusement, en cas de changement on imagine facilement le travail fastidieux que cela génère surtout si ces tests sont simplement des copier-coller qui testent plusieurs cas avec les mêmes messages. Ce qui, en fin de compte, est assez courant dans un développement.

Par contre, on peut garantir que la documentation est bien l'illustration de l'implémentation. Tous les exemples donnés correspondent bien à ce que l'on aura si l'on teste de manière interactive.

Ajoutons que les exemples choisis devraient être avec des valeurs cibles. Ici par exemple nous traitons une fonction qui n'est pas dans le contexte courant de l'interpréteur Python.

```
File "__main__", line ?, in __main__.EvalRedLinda
Failed example:
print "voici le résultat de mon expression", monEval.expressionEvaluated
Expected nothing
Got:
voici le résultat de mon expression 0.551926383871
```

13.3.4 Respect des conventions

Nous avons respecté les conventions d'écriture tout en les adoptant. En fait, nous avons utilisé un mixte entre les conventions utilisées généralement en Java et les programmes développés en Python. Cette pratique est courante dans le monde Python. En effet, le respect stricte des convention de codage n'est utilisé que pour les modules et les bibliothèques standards.

En utiliant l'outil Pylint, nous avons donc défini un fichier de convention que nous annexons.

La contrainte (voir point 12.6 page 80) est donc patiellement atteinte si l'on se place du côté des puristes. Mais elle facilitera la lecture pour tous les programmeurs Java sont les plus nombreux.

13.3.5 Compatibilité syntaxique avec Pylinda

Notre implémentation respecte la syntaxe de Pylinda (cfrn 12.8 page 81).

13.4 Outils

Introduction

Les outils que nous avons utilisés pour la réalisation de ce mémoire sont nombreux. Tous sont open source (cfr point La contrainte : voir point 12.2 page 79). Il témoigne du dynamisme des développeurs ou sociétés qui les mettent à disposition ainsi que de la maturité de ces outils .

13.4.1 Outils de développement

Pour les outils de développement, nous n'avons pas utilisé l'outil de base de Python qu'est Idle mais nous avons préféré nous tourner vers le framewok Eclipse avec les extensions spécifiques à Python. Nous avons aussi utilisé des outils de versionning tels que subversion[10] ou svk[11] que nous avons trouvés plus adaptés à notre situation.

13.4.2 Outils de vérification

Pour les outils de vérification du code, nous avons surtout utilisé py test[12] dont les fonctionnalités ainsi que la stabilité nous a plus. Nous avions aussi essayé Python-Nose [13] mais malheureusement un bug, probablement dû au fait que nous codions en utf8, ne nous a pas permis de l'utiliser. Nous avons aussi enployé l'outil livré en standard avec Python qui est doctest [14].

Mentionnons pour terminer que nous avons utilisé Pylint qui est un outil de contrôle statique[15] afin de vérifier de manière épisodique notre code.

13.5 Améliorations et prospective

[10] http://subversion.tigris.org/
[11] http://www.elixus.org/
[12] http://codespeak.net/py/dist/test/
[13] http://code.google.com/p/python-nose/
[14] http://docs.python.org/library/doctest.html
[15] http://www.logilab.org/857

Chapitre 14

Implémentation

14.0.1 Introduction

Dans ce chapitre nous allons aborder les lignes de force de notre implémentation. Notre intention n'est pas de noyer le lecteur de détails. Les sources étant disponibles, nous avons opté pour cette manière de faire.

14.0.2 Le serveur : Redis

Pour des raisons d'optimisation nous avons dû adapter Redis qui joue ici le rôle de serveur d'espace de tuples. Cette modification est donc faite à deux niveaux :

- au niveau du serveur lui-même écrit en C ;
- au niveau de L'API client écrite en Python.

Notons que cette modification a été facilitée par la qualité aussi bien du code C que de l'API Python.

La commande *keys* de Redis permet d'obtenir la liste des clés qui correspondent aux critères de récherche. Pour notre implémentation, cette commande était seulement adaptée pour les primitives collect et copy_collect. Pour les commandes telles que in, inp, rd, rdp où un seul enregistrement est nécessaire, elle produisait un trafic réseau et un traitement inutiles.

Nous avons donc simplement ajouté la primitive *keysOne* qui retourne un seul élément primitif si celui-ci existe et rien dans le cas contraire.

Pour ce faire nous avons dupliqué la fonction keys et nous l'avons modifiée pour qu'elle retourne le premier élément qu'elle trouve.

Nous avons aussi augmenté le protocole de Redis et fait la liaison entre le protocole et la nouvelle fonction que nous venions de créer.

Quant à l'API, nous avons simplement enrichi le vocabulaire du procole par le *keysOne*.

14.1 Redlinda

14.1.1 Les paramètres de configuration de Redis

Pour pouvoir contacter le serveur Redis les paramètres suivants doivent être initialisés :

- host=None ;
- port=None ;
- timeout=None ;
- db=None ;
- nodelay=None ;
- charset='utf8' ;
- errors='strict'.

C'est la méthode de classe setParamRedis() qui va initialiser les paramètres de connexion. Pour ce faire, nous utilisons un fichier de configuration config.py dont les valeurs sont des variables globales. Ce choix peut être critiqué mais il est possible au programmeur de modifier ces variables membres via des setters. Le programmeur peut aussi récupérer sous forme de liste ces variables.

14.1.2 Algorithme pour rdp

14.1.3 TupleSpace Id Unique

14.1.4 Sérialisation

La sérialisation est obligatoire si nous voulons que la classe soit pickelisable. Mais nous avons quand même un gros souci parce que nous avons un objet sock.

Redis offre la possibilité de stocker des chaînes de caractères, des listes et des ensembles.

Au départ, le choix des listes semblait intéressant. Mais tant la création que l'extraction ne sont pas adaptées à notre implémentation. Lors de l'écriture on ne peut écrire qu'un seul élément à la fois. Cela pose problème puisque le tuple peut être récupéré avant son écriture complète et définitive sur le serveur Redis. Si nous avions choisi cette solution, nous aurions été obligé de mettre un lock lors de l'écriture et de l'enlever lorsque tous les éléments du tuple auraient été stockés sur le serveur Redis.

La récupération d'une liste se fait en trois étapes si on ne connaît par la clé de la liste, ce qui est notre cas. La première étape consiste à récupérer la clé, la seconde à récupérer la longueur de la liste et enfin la troisième à récupérer le contenu de la liste.

Vu ces désavantages, nous avons préféré utiliser le stockage de nos tuples sous forme de chaînes de caractères. Mais comme nous souhaitions conserver le type des éléments de nos tuples, nous créons d'abord une liste des éléments que nous sérialisons, puis nous sérialisons la liste.

Nous utilisons la sérialisation sous forme de chaînes de caractères. Par mesure de sécurité cette chaîne sera elle-même encodée en utilisant "l'url encoding". Malgré la lourdeur de ce traitement les avantages sont intéressants :

- tous les types de Python qui sont sérialisables sont utilisables ;
- la richesse des types offerte par Python est donc conservée.

14.1.5 Locks

Pour la mise en place des verrous, nous avons opté pour une solution semi-évolutive. En effet, on peut choisir le type de fonctionnement de verrou que l'on souhaite obtenir.

Comme Redis ne garantit pas l'atomicité des transactions, nous avons choisi dans tous les cas d'utiliser un verrou transactionnel.

Nous avons aussi choisi d'utiliser un verrou global pour pouvoir protéger les sections critiques.

Pour la mise en place des verrous, nous avons choisi quatre types de verrous :

- verrou sur le serveur Redis ou verrou global ;
- verrou de transaction ;
- verrou sur un TupleSpace ;
- verrou sur la signature d'un tuple ;

- on récupère le tuple sur le serveur ;
- on renomme la clé du tuple ;
- on recrée le tuple sur le serveur ;
- efface la demande sur le serveur ;
- attention on reçoit bien un liste... qui contient des tuples ;
- codage pour le lock ;
- codage pour la requête.

Pour la mise en place des verrous, nous avons choisi trois types de verrous :
- verrou sur le serveur Redis ;
- verrou sur un TupleSpace ;
- verrou sur la signature d'un tuple.

Les clés pour ces trois verrous sont construites comme suit :

- lock :G :R pour le lock global sur Redis ;
- lock :G :TS :idTuple : pour le verrou sur TupleSpace ;
- lock :L :idTuple :signatureTuple :longueurTuple : pour un tuple.

Nous enregistrons la demande sur le serveur si nécessaire.

Dans tous les cas nous devons obtenir un verrou de transaction sinon une exception est générée.

Nous vérifions s'il existe un lock global qui nous bloque.

Nous sommes immunisé contre nos propres locks globaux.

Nous analysons la transaction.

Nous fabriquons la signature du tuple.

C'est dans l'analyse de la requête qu'il faut implémenter la gestion des transactions. Elle agit en fonction de la variable membre qui est self.gestionTransaction. L'analyse de la requête retourne un liste avec deux infos :

- peut-on relâcher le verrou transactionnel [0] ;
- peut-on exécuter la requête [1].

On ne peut jamais avoir en retour l'info [0,0]. En effet dans tous les cas, si nous ne pouvons exécuter une opération, nous devons relâcher le verrou de transaction.

Après analyse de la requête et si l'opération est exécutable, selon l'algorithme qui est choisi, soit

nous relâchons le verrou de transaction soit nous le gardons et nous poursuivons le traitement.

14.1.6 Algorithme rd

Nous avions pensé qu'il suffisait de faire successivement un in puis un out pour obtenir un autre tuple mais cela ne fonctionne pas. Redis retient la connexion et donne la même réponse si la demande est identique. N'oublions pas qu'au départ c'est une cache mémoire ...Même en déconnectant cela ne marche pas non plus. La solution que nous avons trouvée consiste à lui faire croire que c'est un autre tuple. Ce qui dans notre implémentation est facile puisqu'il suffit de changer la signature temps (timeId).

Cette solution est à moitié satisfaisante. En effet, Redis retourne bien de manière aléatoire l'information mais il faudrait analyser l'algorithme qu'utilise Redis pour cette opération.

La sémantique de rd() n'oblige en aucune manière à ce que l'élément soit différent à chaque demande. Mais comme nous avions un serveur qui se rapprochait d'une base de données nous espérions au départ avoir un curseur et ainsi parcourir la liste des éléments qui correspondent à notre requête.

Pour terminer, cette interprétation rd() (opération in et out) n'est valable que si Redis est accédé en exclusivité par l'agent. Dans tous les autres cas on prive les autres agents d'une information qui est pourtant bien présente sur le serveur puisqu'un rd ne peut abstraire une information.

14.1.7 Algorithme copy collect

Pour la copie il nous faut les id des tuples d'origine et de destination. Nous partons du principe que le programmeur passe réellement des tuples qui sont effectivement définis, sinon on arrête le programme en signalant l'erreur.

Les id des TupleSpaces sont fournis pour :
- le TupleSpace d'origine par self._id ;
- le TupleSpace de destination par ts.self._id.

Déplacement avec destruction dans le tuple d'origine vers un nouveau TupleSpace.

Par souci de compatibilité nous devons retourner le nombre de tuples déplacés. En effet, on s'attend à ce que le programmeur exécute une lecture destructive (donc un in). On pourrait lui demander d'exécuter une opération non bloquante mais à ce moment-là, la compatibilité est fortement remise en question.

Déplacement sans destruction dans le tuple d'origine vers un nouveau TupleSpace. Par souci de compatibilité nous devons retourner le nombre de tuples déplacés.

En effet, on s'attend à ce que l'utilisateur exécute une lecture destructive (donc un in). On pourrait lui demander d'exécuter une opération non bloquante mais à ce moment-là, la compatibilité est fortement remise en question.

Donc on interdit le changement sur les tuples. Pour éviter qu'il y ait consommation, on vérifie si le TupleSpace contient les tuples copiés.

14.1.8 Temps d'exécution des directives

Dans les primitives de base de Linda, il existe deux types de directives : celles qui sont bloquantes et qui attendent une réponse positive de présence de la part du gestionnaire de TupleSpaces et celles qui testent la présence d'une information sur le TupleSpace. Dans notre implémentation, nous pouvons gérer le temps imparti pour exécuter une instruction qu'elle soit bloquante ou non. Nous avons aussi la possibilité de savoir si notre directive a pu acquérir un lock de transaction dans le temps imparti. Nous n'avons pas a priori attaché une sémantique particulière à ces possibilités. Mais on pourrait le faire. Par exemple, nous pourrions décider que le programmeur puisse utiliser une directive Linda de test de présence sans vouloir gérer de manière explicite par une boucle. Il suffirait alors de configurer la classe pour que le temps imparti soit choisi par le programmeur par un changement de la variable membre.

14.1.9 Le paramètre tsId

Mais nous allons utiliser une autre sémantique quant à l'identification faite par Pylinda. Le premier nombre indiquera le numéro du serveur Redis. Le deuxième indiquera le numéro du TupleSpace sur le serveur Redis.

En résumé :

- pour le TupleSpace universel son identifiant Pylinda sera "0 !0" ;
- pour les autres tuples, l'identifiant sera un caractère alphanumérique laissé au choix du programmeur ;
- pour l'identifiant Pylinda, c'est le serveur Redis qui attribuera l'tsId du TupleSpace. Pour ce faire le client va incrémenter une clé en utilisant la commande INCR qui est atomique : il récupère la valeur pour enfin retirer son lock s'il existe toujours sinon il doit recommencer la procédure.

14.1.10 Compatibilité

Pour communiquer avec l'espace de tuples qui est ici implémenté via le serveur Redis, cette méthode d'initialisation doit permettre la configuration du client quant à la localisation du serveur. Les paramètres tsid=None et gc=True qu'utilise Pylinda sont en fait des paramètres utilisés en interne par l'implémentation. Ces deux paramètres ne sont pas accessibles au programmeur. On les garde ici uniquement par souci de compatibilité de code.

Le paramètre tsId

Pour Pylinda, lors de la création de la classe, on identifie le serveur (secondaire ou principal) ainsi que les TupleSpaces. Cette identification se fait par deux nombres séparés par deux points. Le premier nombre représente le propriétaire (c'est-à-dire le serveur) et le second le TupleSpace. Le TupleSpace universel qui est accessible à tout le monde est identifié par "0 :0". Or nous utilisons le signe " :" comme séparateur de valeurs dans notre implémentation. Donc il sera codé avec un point d'exclamation. Ce qui donne pour le tuple universel "0 !0". Pour notre part, notre implémentation est totalement différente. Mais nous gardons cette façon de faire. Par souci de compatibilité nous utiliserons la notation modifiée de Pylinda.

Le paramètre gc=True

Pylinda met en place un ramasse-miette (Garbage Collector) pour tous les TupleSpaces à l'exception du TupleSpace universel. Notre implémentation ne gère pas ce type de mécanisme. En effet, c'est l'implémentation de Redis qui gère les connexions (sockets) ainsi que la mémoire.

14.1.11 linda.connect()

Dans l'implémentation de Pylinda, pour accéder à l'espace des tuples, chaque client doit se connecter d'abord à un serveur local. Il n'y a donc besoin que d'un port d'écoute qui est forcément local.

Par contre dans notre implémentation c'est le client qui se connecte directement au serveur Redis qui peut être distant.

Pour ce faire le client doit connaître l'emplacement du serveur Redis. Cela est d'autant plus important qu'il doit avoir accès au TupleSpace universel.

Donc lorsqu'il invoque Linda.universe cet espace doit être initialisé. Pour notre implémentation, il n'est pas nécessaire d'appeler linda.connect().

En effet, c'est lorsque le TupleSpace est créé qu'il reçoit ses informations de configuration.

Quand au TupleSpace universel, il est créé lors de l'import du module Redlinda.

Cette fonction retourne toujours True pour des raisons de compatibilité. Mais au-delà de cela, Redis implémente aussi une fonction connect() qui fait que nous ne nous surchargeons pas puisque c'est bien la fonction connect et non la méthode de classe connect(). En cas de risque de surcharge, nous aurions eu trois choix principaux :

- ne rien faire ;
- recopier le code de la méthode de la classe connect() pour ajouter notre code ;
- créer notre propre méthode connect().

Appeler la méthode connect() de la classe parente par ce code :

- utiliser la méthode parent par le code suivant parent = super(TupleSpace,self) parent.connect() ;
- puisqu'en Python l'appel de type super(class, instance) renvoie un objet qui donne accès aux méthodes de la classe supérieure ;
- nous pourrions aussi utiliser cette fonction pour créer un id de client, nous préférons ne pas le faire ici, mais plutôt utiliser les variables globales.

14.1.12 Mode de stockage des tuples

Ecriture d'un tuple dans le TupleSpace. Les tuples sont stockés sur le serveur Redis sous forme de clés-valeurs (keys-values).

Codage de la clé

La clé est constituée d'un ensemble de valeurs séparées par deux points.

Voici les valeurs qui seront présentes dans la clé :

- idPylinda = identifiant utilisé par Pylinda modifié ;

- hash(tuple) = Ce hash est obtenu après avoir concaténé le type de chaque composant du tuple ;
- len(tuple) = indique le nombre d'éléments du tuple ;
- data = une liste qui contient les valeurs de la clé ;
- hasard+time = tentative de générer un nombre aléatoire nous devrions utiliser une bibliothèque spécialisée ou un système de hash.

Redis ne supporte pas les espaces dans les clés. C'est pourquoi la clé sera transformée en utilisant l'*url encoding*. L'*url encoding* est bien supporté par les serveurs en général et par Redis en particulier.

Remarque sur le codage des datas. L'utilisation en tant que tel des datas pour construire la clé est assez naïve. Elle part du principe que les données seront des nombres ou des chaînes de longueur raisonnable. De plus comme nous utilisons un séparateur " :", elles ne doivent pas contenir ce caractère.

La première version qui implémente cette vision devra être corrigée dans la seconde version en utilisant un hash des données.

Des précautions supplémentaires devront être prises pour vérifier que le tuple récupéré correspond bien au template.

En effet, en cas d'utilisation de hash, on n'est pas à l'abri d'un télescopage.

Pour valider cette solution nous ferons des tests :

- avec le hash ;
- sans hash.

Codage des datas

Nous avons décidé de sérialisé les datas. C'est une première opération qui est faite. Par la suite, comme nous le verrons dans la partie labo, nous testerons la compression des données. Pour ces deux points, voir la sérialisation des données et la compressions des données.

14.1.13 La compression des données

14.1.14 Le verrou global

Nous avons donné la possibilité au programmeur de protéger des parties critiques de son code ou d'exécuter des directives plusieurs lignes, tout en s'assurant de la cohérence des données. Par exemple : si la donnée a existe sur le TupleSpace A :

- nous consommons la donnée a ;
- nous copions les tuples(int, int) du TupleSpace A vers le TupleSpace B.

En effet, la sémantique de ce code pourrait être que l'on crée des ensembles qui doivent être traités de manière holistique.

Pour ce type de besoin, le programmeur réserve le serveur Rédis le temps qu'il jugera nécessaire.

14.1.15 Optimisation du code

Opération gourmande

Dans un souci de rapidité, nous avons choisi que le client fasse les opérations gourmandes en temps avant de prendre contact avec le serveur quitte à ce que cela soit superflu.

Duplication de code

Dans certains cas, nous avons préféré ne pas créer de méthode membre pour accélérer le traitement. En effet, l'appel à une fonction ou à une procédure est coûteux en temps. Mais dans un souci de maintenance du code, nous avons été parcimonieux avec cette méthode.

Compression des données

Cette partie sera testée dans la quatrième partie : le labo.

14.1.16 Les tuples et les patrons

Pour récupérer le tuple, nous avons besoin de trois types d'information qui sont implicites à notre implémentation :

- le pyld du tuple ;
- le hash de la signature du tuple ;
- la longueur du tuple,

ainsi que le type d'information que le programmeur formule dans sa requête. Remarquons que timeld n'est pas utilisé car il sert en quelque sorte de hash.

Remarquons que les instances sont simplement codées dans la clé avec le mot clé instance. En effet, les seules instances qui nous intéressent ici sont l'objet TupleSpace. Il serait possible de stocker d'autres types d'objet avec le serveur Redis. Il faudrait s'assurer alors d'obtenir un identificateur unique auprès du serveur Redis. Une autre manière de faire serait de récupérer une instance et d'utiliser l'introspection pour s'assurer que l'objet est bien celui que l'on attend. Cette dernière solution est quand même très hasardeuse. Attention, les instances TupleSpaces sont "printables" car elles possèdent un attribut __str__. Mais nous n'utilisons pas cette fonctionnalité ici.

Remarquons aussi que les types des templates sont toujours évalués au niveau du type après évaluation de l'expression.

Donc a=["un",bool,3,rouge+1] est bien évalué comme suit :

- "un" -> str : string
- bool -> <type bool> : un type booleen
- 3 ->int : un entier
- rouge+1 ->int : si rouge est int alors la somme est int. Notre implémentation ne considère pas qu'il s'agit d'un type somme. Même si comme nous le verrons par la suite nous avons donné la possibilité d'utiliser un tuple vivant.

Notre implémentation différencie les <type de ...> et les types. Quand on désigne directement un type, il est remplacé par star dans la recherche.

<type 'instance'> est renvoyé pour une classe. Attention c'est un choix d'implémentation de ne tenir compte que du type d'instance.

Encode chaque élément du tuple. Ceci est utilisé car Redis ne supporte pas les espaces et autres caractères dans les keys. Nous utilisons ici l'url "encoding" qui propose une méthode reconnue par la majorité des navigateurs webs et des serveurs.

Pour différencier les tuples entre eux qui sont produits par un même client, nous utilisons la concaténation du temps machine et d'un nombre au hasard. Attention c'est un pseudo hasard. Nous nous demandons si nous ne devrions pas simplement demander un nombre unique au serveur Redis.

[fixme] : le mieux c'est de faire appel à une bibliothèque spécialisée.

14.2 Les Classes d'erreurs

Pour communiquer avec l'extérieur, notre classe utilise une série de messages. Nous avons voulu que ceux-ci soient particuliers à chaque situation. Cela permet deux choses : la première de pouvoir tester une classe en particulier en changeant les paramètres de RedLinda, la seconde de pouvoir être plus précis au niveau du message. Nous avons opté pour l'arrêt dans un état stable du programme en cas d'erreur. Nous avons fait ce choix car nous estimons qu'il est important que les erreurs soient bien visibles. Rappelons que nous avons voulu construire une boite à outils pour expérimenter et non un logiciel de production.

Ces classes d'erreurs se répartissent en cinq catégories :

- la gestion des locks (acquisition, relaxe) ;
- le temps imparti à une directive bloquante pour s'exécuter ;
- le temps imparti à une directive non bloquante pour s'exécuter ;
- l'utilisation d'une fonction non encore implémentée ;
- la gestion des opérations en cours pour la classe.

14.3 Tuple vivant

14.3.1 EvalRedLinda

Cette classe implémente un tuple vivant.

Pour ce faire nous avons la richesse de Python. En effet, celui-ci permet d'interpréter du code Python dans un programme Python.

Python met principalement à disposition deux instructions (exec et execfile) et une fonction (eval). L'exécution peut se faire soit en évaluant une chaîne (comme un programme d'ailleurs), soit en exécutant du code déjà interprété. Dans ce dernier cas on utilise la fonction compile() qui retourne un objet code. Ce code objet est différent pour eval() ou pour exec[file](). Un paramètre permet d'indiquer à compile le client final.

Dans notre implémentation, le type de l'élément vivant du tuple est donc avant son évaluation de type EvalRedLinda (un type class donc). Après son évaluation il peut être d'un autre type.

Cette évaluation peut prendre différentes formes puisque le langage Python l'autorise. Cela peut être soit une évaluation d'une expression ou d'un code, soit l'exécution d'un bout de code ou d'un

fichier de code.

Donc l'expression à évaluer se présente sous deux formes :

- la forme fonctionnelle cos (90) +cos (60) ;
- la forme instruction a= 10+20. Cette dernière peut être un nombre n de lignes.

14.3.2 La forme fonctionnelle (eval)

La forme fonctionnelle est implémentée avec l'instruction eval() de Python. C'est une fonction qui retourne un résultat que nous pouvons directement affecter à une variable. Le contexte d'exécution est le même contexte que le reste du programme. Cela simplifie évidemment les choses puisque nous pouvons utiliser directement les bibliothèques, les objets ainsi que toutes les variables globales. Si nous voulons isoler ce nouvel appel, deux choix s'offrent à nous. Soit nous lui passons une copie du scope actuel, ce qui est de loin la solution la plus facile, soit nous sommes obligé de lui passer un objet compilé dans le cas où nous voudrions utiliser une bibliothèque non présente par défaut dans l'interpréteur Python. Dans ce dernier cas nous devons faire appel à la méthode compile pour préparer ce code pour l'expression eval.

Pour notre part, nous avons choisi de toujours utiliser le contexte actuel.

14.3.3 La forme instruction (exec)

Cette forme est à proscrire car elle n'est pas le signe d'une bonne pratique de programmation. En effet, elle se résume à l'exécution d'un programme Python à l'intérieur d'un programme Python. En tout premier lieu, il est nécessaire de lui créer un nouveau contexte. En effet, si ce bout de code réimporte une bibliothèque déjà présente, des erreurs risquent de se produire. Réinitialisation intempestive par exemple. Le nouveau contexte est accessible par un dictionnaire. Signalons que le débogage de ce type de cette manière de faire est assez délicat puisque soit le code s'exécute dans un contexte isolé et nous n'utilisons pas le contexte soit nous récupérons ce contexte et à ce moment-là la démarche fonctionnelle est plus simple.

Pour notre part nous n'utiliserons pas ce type d'approche.

14.3.4 Cas d'utilisation

Voir la section mode d'emploi

14.3.5 L'implémentation proprement dite

Le mécanisme que nous utilisons est le suivant : lors de l'enregistrement de l'élément vivant dans le TupleSpace, l'expression n'est pas évaluée. Cette évaluation sera exécutée lors de sa lecture ou retrait du TupleSpace.

Tous nos objets sont sérialisés lors de leur stockage sur le TupleSpace. Lors de leur désérialisation nous appelons une seconde fois la méthode init qui va provoquer l'interprétation de la variable membre expression.

Pour mettre en place ce mécanisme nous avons besoin d'un drapeau evalLiveTuple qui est faux lors de l'initialisation de la classe et prend l'état de Vrai après initialisation.

Lors de la seconde initialisation c'est ce drapeau qui provoque l'évaluation. Après évaluation la variable membre expression contient toujours l'expression à évaluer. Cette information permet un contrôle.

Cette classe utilise les paramètres suivants :
- **expression** : qui contient soit l'instruction soit la fonction a évaluer ;
- **isFile=False** : qui indique si l'expression est un fichier ;
- **typeEval=True** : qui est à Vrai pour la forme fonctionnelle et faux pour la forme instruction ;
- **globalDico=None** : qui permet de définir le contexte global (scope) ;
- **localDico=None** : qui permet de définir le contexte global (scope) ;
- **evalLiveTuple=False** : qui force l'évaluation de l'expression.

Après évaluation de l'expression, les valeurs sont accessibles de manière différente selon la forme de l'expression :

- **la forme fonctionnelle** : la valeur se trouve dans la variable membre expressionEvaluated. Le contexte est par défaut le contexte courant. Mais soulignons toutefois que ce contexte n'est pas accessible dans le scope global. En effet, l'évaluation se fait bien au niveau de la classe et non au niveau global ;
- **la forme instruction** : la ou les valeurs se trouvent dans les deux dictionnaires des contextes des variables membres globalDico et localDico. C'est le dictionnaire localDico qui contient en fait le contexte global de l'exécution alors que globalDico contient le contexte global lors de l'appel de l'exécution. Nous retrouverons donc nos valeurs dans le dictionnaire localDico. Attirons l'attention que seules les valeurs connues au premier niveau d'exécution sont accessibles. On peut comprendre dès lors que le débogage est une véritable partie de plaisir puisque nous avons en fait une boite noire.

Remarque importante :

L'évaluation se fait dans le contexte que l'on passe à l'initialisation de la classe. Si aucun contexte n'est passé, c'est avec un dictionnaire vide pour le contexte que la classe est initialisée. Sans cela, il est impossible d'avoir des valeurs manipulées par le code exécuté lors de l'évaluation.

Cet aspect des choses fait qu'il vaut mieux considérer ce type d'utilisation comme une "procédure à l'ancienne". Même si le langage Python ne fait pas la différence.

Il est possible de faire une copie du contexte courant si on le souhaite. Cette manière de faire est une très mauvaise idée comme nous l'avons souligné précédemment. La duplication d'import de classe est source d'erreurs.

Il vaut mieux donc d'isoler le plus possible le code que l'on va exécuter via cette stratégie.

En conclusion

Notre classe EvalRedLinda implémente les deux manières de faire relatives à l'évaluation d'une expression de Python. Que l'on veuille utiliser exec ou eval, ce n'est pas à nous à restreindre la richesse de Python. En effet, notre classe ne fait qu'ajouter une couche d'abstraction pour permette à RedLinda de proposer les tuples vivants.

14.4 Les fichiers du module Redlinda

14.4.1 Arborescence

```
|-- __init__.py
|-- config.py
|-- evalredlinda.py
|-- kernel.py
|-- redis.py
|-- utils.py
|-- tests
|   |-- test_erreur.py
|   `-- test_primitive.py
```

14.4.2 __init__.py

Ce fichier initilisalise simplement le module Redlinda. Il est obligatoire pour la gestion des packages dans l'environnement Python.

14.4.3 config.py

Ce fichier permet de configurer les paramètres que va utiliser la classe RedLinda pour communiquer avec le serveur Redis.

14.4.4 evalredlinda.py

Ce fichier définit la classe EvalRedLinda. Il incorpore également les tests de non régression basés sur la documentation puisque c'est ce choix que nous avons fait.

14.4.5 kernel.py

Ce fichier est le coeur de Redlinda. En effet, il définit deux classes :

- la classe RedLinda ;
- les différentes classes d'erreurs qu'utilise RedLinda pour signaler les problèmes d'exécution.

14.4.6 redis.py

Ce fichier est fourni par le projet Redis. Il permet d'implémenter un client en Python pour le serveur Redis. Nous avons simplement ajouté la définition d'une nouvelle fonction membre pour les besoins de l'implémentation de Redlinda.

14.4.7 utils.py

Ce fichier comporte plusieurs fonctions utiles à la classe RedLinda. On peut y trouver par exemple la sérialisation des données ou le codage des clés d'index qu'utilise la classe RedLinda.

14.4.8 test_erreur.py

Ce fichier permet de faire les tests de non régression pour la classe RedLinda.

14.4.9 test_primitive.py

Ce fichier permet de faire les tests de non régression pour toutes les classes d'erreurs de Redlinda.

Chapitre 15

Mode d'emploi et exemple

15.1 Redlinda

15.2 Tuple vivant

Pour illustrer le mode d'emploi de cette classe, nous allons présenter une session interactive en la commentant.

Instanciation

Instanciation d'une variable de classe EvalRedLinda.

```
>>> evalOne=None
>>> evalOne=EvalRedLinda("somme = 10+20")
>>> print "voici le type de evalOne", type(evalOne)
voici le type de evalOne <class '__main__.EvalRedLinda'>
```

Un session interactive exec

Ici nous allons commencer par tester la forme de l'instruction en prenant l'instruction exec comme exemple. Comme cela est illustré dans le code, nous utilisons un nouveau contexte en déclarant un dictionnaire pour notre scope global. Après exécution, nous avons accès à toutes les variables du contexte d'exécution via le dictionnaire monDico. Ici la variable somme est accessible via le dictionnaire que nous avons passé en argument à notre instruction exec.

```
>>> monDico={}
>>> exec(evalOne.expression,monDico)
>>> # pour accéder à la variable résultat

>>> print "voici l'évaluation de la variable somme ",monDico['somme']
voici l'évaluation de la variable somme  30
```

Voici maintenant la même instruction qui s'applique ici à un fichier tableMultiplication.py qui est un fichier Python dont voici le contenu :

```
from math import *

unNombre= 12
uneListe =[]
for compteur in xrange(0,unNombre+1):
    uneListe.append(compteur * unNombre)

monSinus = sin(90)
```

Voici la session interactive :

```
>>> # utilisation de l'instruction execfile

>>> execfile("tableMutiplication.py",monDico)
>>> print "voici la liste", monDico['uneListe']
voici la liste [0, 12, 24, 36, 48, 60, 72, 84, 96, \\
                    108, 120, 132, 144]
>>> print "voici le sinus", monDico['monSinus']
voici le sinus 0.893996663601
```

Comme nous pouvons le constater, cette dernière manière de faire n'est pas vraiment souple par rapport à l'orienté objets. Il n'est vraiment pas recommandable d'utiliser cette manière de faire.

Voici enfin la manière fonctionnelle. L'instruction est d'abord exécutée dans un nouveau contexte en dupliquant simplement le contexte courant à l'aide de la méthode **globals**.

```
>>> # utilisation dans un nouveau contexte d'exécution
>>> evalOne = None
>>> evalOne = EvalRedLinda("cos(90)+cos(0)")
>>> monDico=globals()
>>> resultatExpression = eval(evalOne.expression,monDico)
>>> print "voici le résultat de mon expression",resultatExpression
voici le résultat de mon expression 0.551926383871
```

L'instruction est ensuite exécutée dans le contexte courant. Cette manière de faire est en définitive la plus souple et permet un meilleur contrôle de ce que l'on fait. Pour ces raisons nous utiliserons principalement cette méthode.

```
>>> evalOne.expression="cos(90)+cos(0)"
```

```
>>> # utilisation de la fonction eval dans le même scope

>>> resultatExpression=eval(evalOne.expression)
>>> print "voici le résultat de mon expression",resultatExpression
voici le résultat de mon expression 0.551926383871
```

Quatrième partie

labo

introduction

Dans cette partie, nous avons confronté notre implémentation au traitement du son et de l'image. Nous avons grâce, aux expériences, pu améliorer notre implémentation. Cette partie, loin d'être exhaustive sur les tests que nous avons réalisés veut simplement proposer quelques points.

Chapitre 16

Notre environnement

Introduction

Face à la diversité des possibilités, nous avons opté pour certains outils. Précisons que même si ces choix sont justifiés, il existe une part d'arbitraire dans chaque choix. En effet, pour des raisons pratiques, nous avons marqué notre préférence pour des outils que nous utilisons dans le cadre de notre travail quotidien. Il nous paraissait inutile de réapprendre la roue.

16.1 Son-image

Nous avons utilisé des librairies spécialisées qui nous permettaient de tester rapidement grâce à leur niveau d'abstraction des traitements parfois complexes. Parmi ces outils citons :

- PIL (Python Image Library) permet de découvrir le domaine mais son ignorance pour les matrices le rend très lent pour les calculs. L'avantage qu'il possède c'est qu'il est simple à prendre en main et se présente tout compte fait comme un bon couteau suisse.
- Vips qui est une bibliothèque C++ qui offre une API Python. Cette bibliothèque est meilleure et surtout plus rapide que PIL (Python Image Library)
- scipy qui est une bibliothèque scientifique de très haute qualité. Elle offre des outils à la fois complets et relativement rapides. L'implémentation du calcul matriciel, par exemple, est fortement optimisé.
- audio-lab : Audio-lab est une couche d'abstraction de la bibliothèque scipy. Il permet en plus de manipuler des matrices grâce à numpy. De ce fait, il s'avère un choix de qualité en conjugaison avec scipy.
- Sox : que nous avons bien entendu utilisé est un outils très polyvalent et qui permet surtout de comprendre facilement les concepts qu'il manipule. Son usage, à premier vue austère, permet par exemple en un tour de main de convertir d'un format à un autre.

16.2 Notre matériel

Nous avons fait les tests avec :

- Notre parc machine était constitué :
  - de deux tours (pentium 4 et double coeurs à 3 Ghz)

- d'un portable (pentium centrino 1.6)
- un switch 100/1000 de moyenne gamme mais qui nous a permis de tester facilement les goulets d'étranglement et les déconnexions d'agent.
- Un nas gigabit (mais le processeur arm ne pouvait pas fournir un débit assez puissant.

16.3 Système d'exploitation

Nous avons utilisé exclusivement Linux (Distribution Debain Like)

16.4 Coordination

Pour les langages de coordination, nous avons travaillé uniquement avec :

- Pylinda ;
- Redlinda (notre implémentation) ;

16.4.1 Architecture Logicielle

Nous avons exclusivement testé l'architecture Master-Worker.

Chapitre 17

Les tests

17.1 La compression

Pour la compression, nous avons réussi à obtenir un taux de compression de 10 pourcent.

Ce pourcentage très faible, nous le devons à notre choix de Redis. En effet, celui-ci n'est prévu que pour manipuler des clés-valeurs et non du binaire.

Lors de la compression, nous devons donc transformer notre code binaire en chaîne compatible pour redis soit de l'UTF8 ou de l'*url encoding*.

Nous pensions obtenir de meilleurs taux de compression en utilisant non plus l'*url encoding* mais bien Base64.

Nous n'avons pas implémenter le codage Base64 car il nous a fallu analyser pourquoi notre codage binaire ne passait pas.

17.2 Le hash des données dans la clé

Lors de la première implémentation, nous avions fait le choix de coder *naïvement* les datas dans la clé. Cela fonctionnait parfaitement pour des tuples de petites tailles. Lorsque nous avons aborder le traitement du son et de l'image, nous nous sommes rendu compte que cette stratégie était un véritable gouffre mémoire.

En adoptant une stratégie de hash, nous avons pu réduire fortement la consommation mémoire et traiter ainsi des blocs de données plus conséquents.

17.3 Comparaison Redlinda et Pylinda

17.3.1 Vitesse

Notre implémentation est plus rapide que celle de Pylinda. Cela s'explique par deux raisons principales :

- notre serveur de tuple est un serveur en C et il est optimisé ;

– nous utilisons la compression des données et non Pylinda ;

17.3.2 Sémantique

Comme nous l'avons signalé, nous ne détectons pas les deadlocks. Mais il est possible de l'implémenter via un programme externe.

Notre implémentation, par contre permet en plus de définir des zones critiques, ce qui, nous croyons, est un véritables point fort. Cela permet aussi de diminuer les deadlocks.

Chapitre 18

Les traitements choisis

Introduction

Les traitement choisis se basent tous sur le traitement du signal. Nous n'envisageons que ce type de traitement en écartant par exemple le traitement sémantique.

18.1 Son

Pour le son, nous avons utilisé une DCT. Pour ne pas noyer les worker et surtout le serveur d'espace de tuple, nous avons du mettre une temporisation. Ce qui nous amène à la conclusion que le master ne doit pas suivre la cadence sous peine de voir l'espace de tuple s'effondrer.

18.2 Image

Pour cette partie, nous mettons en annexe un programme qui transforme une image à trois canaux RVB en nuance de gris selon la norme *ITU-R 601-2 luma transform :*

Nous fournissons d'abord la classe qui implémente ce traitement. Ensuite, à partir d'un template d'un Master et d'un Worker, nous distribuons la classe que nous avons implémentée.

Ce programme est un bon exemple de la bande passante qu'il faut et de la puissance utilisée pour faire un simple traitement. Pour optimiser ce programme, les stratégies à mettre en place sont vraiment très nombreuses.

Chapitre 19

Analyse des résultats

Le gain de traitement en utilisant une implémentation du langage Linda est relativement inexistant si l'échange des données se fait via l'espace de tuple et que les données passent par le réseau. Par contre, si chaque Worker dispose d'un dépôt local, la qualité de synchronisation et donc les gains de productivité sont évidents. Les ordinateurs sont toujours à leur maximum de traitement, puisque la synchronisation se fait naturellement.

Cette même remarque s'impose dans le cas d'un bus de données très rapide comme par exemple sur une machine multiprocesseurs ou sur une machine multicoeurs.

Conclusions

Les efforts à faire pour optimiser un langage de coordination et pour l'utiliser dans le cadre du traitement de l'image ou du son sont importants. Mais la facilité de coordination qu'apporte un langage de type Linda est vraiment impressionnante.

Conclusions générales

L'objectif de ce mémoire était d'implémenter un langage de coordination et de le confronter à deux domaines gourmands en temps processeur. Nous avons choisi le traitement du son et de l'image.

L'implémentation d'un langage de coordination *à la Linda,* fut pour nous une expérience passionnante.

Au travers de la présentation de Linda dans la première partie, nous avons montré que la diversité amenée par un manque de précision sémantique des primitives, permettait des choix certains lors d'une implémentation.

Les quatre implémentations que nous avons présentées dans la seconde partie, témoignent bien de ce choix.

Dans l'implémentation, nous avons nous-mêmes été confronté à ces choix. Ces derniers sont bien réels. Nous avons remarqué que malgré une analyse sérieuse, il restait toujours plusieurs possibles.

Au terme de ce mémoire, il apparaît qu'un langage de coordination doit d'une part définir clairement son modèle pour que ses primitives puissent être implémentées sans ambiguïté.

Il apparaît aussi qu'il faut fournir avec ce paradigme qui s'avère complexe des outils de monitoring et de débogage hors pair.

Comme nous l'avons constaté dans la partie labo, il est très difficile, étant donné le flot d'informations qui circule à travers le réseau de savoir où se trouvent les goulets d'étranglement. Ajoutons que le côté asynchrone du modèle ne permet pas facilement de détecter les deadlocks ou autres erreurs de conception.

Nous constatons que la programmation parallèle et distribuée demande non seulement des modèles efficaces au niveau de la synchronisation mais aussi des modèles efficaces à tous les niveaux de notre métier d'architecte de logiciels.

Bibliographie

[Académie Française,] ACADÉMIE FRANÇAISE. Dictionnaire de l'académie française. Site internet `http://www.cnrtl.fr/definition/academie9/image`. Page accédée le 4 juin 2009.

[Accord, 2002] ACCORD, P. (2002). Etat de l'art sur les langages de coordination. Rapport technique, Projet Accord.

[Adobe, 1992] ADOBE (1992). Tiff : revision 6.0. Adobe Developpers Association Site Internet `http://www.adobe.com/Support/TechnNotes.html`. Page accédée le 15 août 2009.

[Bellanger, 2002] BELLANGER, M. (2002). *Traitement numérique du signal. Théorie et pratique*. Dunod.

[Berry et Boudol, 1992] BERRY, G. et BOUDOL, G. (1992). The chemical abstract machine. *Theoretical Computer Science*, 96(1):217–248,.

[Butcher *et al.*, 1994] BUTCHER, P., WOOD, A. et ATKINS, M. (1994). Global synchronisation in linda. *Concurrrency : Practice and Experience*, 6(6):505–516.

[Carriero et Gelernter, 1989] CARRIERO, N. et GELERNTER, D. (1989). Linda in context. *Communications of ACM*, 32:444–458.

[Carriero et Gelernter, 1990] CARRIERO, N. et GELERNTER, D. (1990). *How to write parrallel programs, A first course*. Mit Press, cambridge.

[Cottet, 1997] COTTET, F. (1997). *Traitement des signaux et acquisition de données. Cours et exercicesrésolus*. Dunod.

[Dijkstra, 1971] DIJKSTRA, E. W. (1971). Hierarchical ordering of sequential processes. *Acta Informatica*, 1:115–138.

[Ericsson-Zenith, 1992] ERICSSON-ZENITH, S. (1992). *Process Interaction Models*. Thèse de doctorat, UNIVERSITE PARIS VI.

[Gelernter et Carriero, 1992] GELERNTER, D. et CARRIERO, N. (1992). Coordination languages and their significance. *Communications of ACM*, 35:96–107.

[Gilbert, 2009] GILBERT, V. (2009). *Développer ses fichiers RAW*. 3 édition.

[Gonzalez et Woods, 2002] GONZALEZ, R. C. et WOODS, R. E. (2002). *Digital Image Processing*. 2ème édition édition.

[Hoare, 1978] HOARE, C. R. (1978). Communicating sequentials processes. *CACM*, 21(8):666–677.

[IPTC, 2009] IPTC (2009). International press telecommunications council. Site internet `http://fr.wikipedia.org/wiki/International_Press_Telecommunications_Council`. Page accédée le 21 août 2009, page modifiée le 19 juin 2009.

[Jacob et Wood, 2000] JACOB, J. L. et WOOD, A. M. (2000). A principled semantics for inp. *Lecture Notes In Computer Science, Proceedings of the 4th International Conference on Coordination Languages and Models*, 1906:51–65.

[Kurtnoise, 2004] KURTNOISE (2004). Les formats audio lossless. Site Internet : `http://www.unite-video.com/phpbb/viewtopic.php?p=55157`. Page mise à jour le 4 septembre 2004, page accédée le 8 avril 2009.

[Omicini *et al.*, 2001] OMICINI, A., ZAMBONELLI, F., KLUSH, M. et TOLKSDORF, R., éditeurs (2001). *Coordination of Internet Agents : Models, Technologies, and Applications.* Springer.

[Papadopoulos et Arbab, 1998] PAPADOPOULOS, G. A. et ARBAB, F. (1998). Coordination models and langages. *Software Engineering (SEN)*, SEN-R9834.

[Printz, 2009] PRINTZ, J. (2009). *Architecture logicielle : concevoir des applications simples, sûre et adaptables.* Dunod, 2ème édition édition.

[Rota, 2009] ROTA, V. M. (2009). *Gestion de projet Vers les méthodes agiles.* 2ème édition.

[Rowstron et Wood, 1996] ROWSTRON, A. et WOOD, A. (1996). Solving the linda multiple rd problem. *LNCS*, 1061:357–367.

[Schumacher *et al.*, 1998] SCHUMACHER, M., KRONE, O., CHANTEMARGUE, F. et HISRBRUNNER, . (1998). Stl : Un modèle et langage de coordination pour les systèmes distribués. Site Internet : `http://liawww.epfl.ch/~schumach/publications/schumacher_RENPAR10_1998.PDF`. Actes des Rencontres Francophones deu Parallélisme des Architectures et des Systèmes. Strasbourg, France 9-12 juin 1998.

[Scott, 2003] SCOTT, W. (2003). Wave pcm soundfile format. Site Internet : `http://ccrma.stanford.edu/courses/422/projects/WaveFormat/`. Page mise à jour le 20 janvier 2003, page accédée le 26 février 2009.

[Swinnen, 2009] SWINNEN, G. (2009). *Apprendre à programmer avec Python.*

[Wikipedia, 2009a] WIKIPEDIA (2009a). Codage informatique des couleurs. Site internet : `http://fr.wikipedia.org/wiki/Codage_informatique_des_couleurs`. Page accédée le 25 mai 2009, page modifiée le 17 mai 2009.

[Wikipedia, 2009b] WIKIPEDIA (2009b). Distributed computing. Site Internet : `http://en.wikipedia.org/wiki/Distributed_computing`. Page mise à jour le 13 août 2009, page accédée le 15 août 2009.

[Wikipedia, 2009c] WIKIPEDIA (2009c). Extensible metadata platform. site internet : `http://fr.wikipedia.org/wiki/Extensible_Metadata_Platform`. Page accédée le 14 juin 2009, page modifiée 9 juin 2009.

[Wikipedia, 2009d] WIKIPEDIA (2009d). Free lossless audio codec - wikipédia. Site Internet : `http://fr.wikipedia.org/wiki/Free_Lossless_Audio_Codec`. Page mise à jour le 12 mars 2009, page accédée le 8 avril 2009.

[Wikipedia, 2009e] WIKIPEDIA (2009e). Fréquence-wikipedia. `http://fr.wikipedia.org/wiki/Fr%C3%A9quence`. Page mise à jour le 17 juin 2009, page accédée le 1 septembre 2009.

[Wikipedia, 2009f] WIKIPEDIA (2009f). Pixel. site internet `http://fr.wikipedia.org/wiki/Picture_Element`. page accédée le jeudi 04 juin 2009, page modifiée le 4 février 2009.

[Wikipedia, 2009g] WIKIPEDIA (2009g). Tagged image file format. site internet `http://fr.wikipedia.org/wiki/Tagged_Image_File_Format`. page accédée le jeudi 11 juin 2009, page modifiée le 25 avril 2009 à 14 :41.

[Wikipedia, 2009h] WIKIPEDIA (2009h). Trie. `http://en.wikipedia.org/wiki/Trie`. Page mise à jour le 13 août 2009, page accédée le 29 août 2009.

[Wikipedia, 2009i] WIKIPEDIA (2009i). Waveform audio format - wikipédia. Site Internet : `http://fr.wikipedia.org/wiki/WAVEform_audio_format`. page mise à jour le 1 février 2009, page accédée le 29 février 2009.

[Wikipedia, 2009j] WIKIPEDIA (2009j). Wavpack - wikipédia. Site Internet : `http://fr.wikipedia.org/wiki/WavPack`. Page mise à jour le 30 mars 2009, Page accédée le 8 avril 2009.

[Zaffalon, 2007] ZAFFALON, L. (2007). *Programmation concurrente et temps réels avec Java.* Presses Polytechniques et universitaires romandes.

[Ziadé, 2006] ZIADÉ, T. (2006). *Programmation Python : Syntaxe, conception et optimisation.*

[Ziadé, 2007] ZIADÉ, T. (2007). *Python Petit guide à l'usage du développeur agile.*

Annexe A

Signal

A.0.1 La DCT

Voici une implémentation naîve de la DCT. La matrice de qualité est définie par la variable qmatrix.

```
from numpy import fft,array,arange,zeros,dot,transpose
from math import sqrt,cos,pi
from numpy import fft,array,arange,zeros,dot,transpose
from math import sqrt,cos,pi
#        """"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""3


qmatrix = [[ 16 , 11 , 10 , 16 , 24  , 40  , 51  , 61 ],
           [ 12 , 12 , 14 , 19 , 26  , 58  , 60  , 55 ],
           [ 14 , 13 , 16 , 24 , 40  , 57  , 69  , 56 ],
           [ 14 , 17 , 22 , 29 , 51  , 87  , 80  , 62 ],
           [ 18 , 22 , 37 , 56 , 68  , 109 , 103 , 77 ],
           [ 24 , 35 , 55 , 64 , 81  , 104 , 113 , 92 ],
           [ 49 , 64 , 78 , 87 , 103 , 121 , 120 , 101],
           [ 72 , 92 , 95 , 98 , 112 , 100 , 103 , 99 ] ]

class DCT(object):
    def __init__( self ):
        pass

    def __transKernel(self,N):
        A = zeros((N,N))
        for x in xrange(0,N):
            for u in xrange(0,N):
                if u==0:
                    A[x][u] = sqrt(1/ float (N))
                else:
                    A[x][u] = sqrt(2/ float (N))*cos(pi*(2*x+1)*u/float(2*N))
        return A

    def __itransKernel( self , N):
        A = zeros((N,N))
        for x in xrange(0,N):
            for u in xrange(0,N):
                if x==0:
                    A[x][u] = sqrt(1/ float (N))
                else:
                    A[x][u] = sqrt(2/ float (N))*cos(pi*(2*u+1)*x/float(2*N))
        return A

    def transform( self , m):
        tk = self .__transKernel(len(m))
        t1 = dot(m,tk)
        t1 = transpose(t1)
        t1 = dot(t1 , tk)
        return transpose(t1)
```

```
    def itransform(self,m):
        tk = self.__itransKernel(len(m))
        t1 = dot(m,tk)
        t1 = transpose(t1)
        t1 = dot(t1,tk)
        return transpose(t1)

    dct_test = [ [-76, -73, -67, -62, -58, -67, -64, -55],
                 [ -65, -69, -73, -38, -19, -43, -59, -56],
                 [-66, -69, -60, -15, 16, -24, -62, -55],
                 [-65, -70, -57, -6, 26, -22, -58, -59],
                 [-61, -67, -60, -24, -2, -40, -60, -58],
                 [-49, -63, -68, -58, -51, -60, -70, -53],
                 [-43, -57, -64, -69, -73, -67, -63, -45],
                 [-41, -49, -59, -60, -63, -52, -50, -34] ]


    idct_test = [ [-416, -33, -60, 32, 48, -40, 0, 0],
                  [ 0, -24, -56, 19, 26, 0, 0, 0],
                  [ -42, 13, 80, -24, -40, 0, 0, 0],
                  [ -56, 17, 44, -29, 0, 0, 0, 0],
                  [ 18, 0, 0, 0, 0, 0, 0, 0],
                  [ 0, 0, 0, 0, 0, 0, 0, 0],
                  [ 0, 0, 0, 0, 0, 0, 0, 0],
                  [ 0, 0, 0, 0, 0, 0, 0, 0] ]

    if __name__=="__main__":
        dct = DCT(2)
        print "testing transform..."
        print dct.transform(dct_test)
        print "testing itransform..."
        print dct.itransform(idct_test)
```

A.0.2 Les filtres à réponse impulsionnelle finie

Voici en langage Python un filtre RIF :

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""
Implémentation d'un filtre à réponse impulsitionnelle finie.
Cette implémentaton se base sur le type array de numpy.

Pour verifier son implémentation, j'utilise signal. lfilter  de scipy.
De ce fait, j' utilise  la même notation pour les coffecients a et b.

TODO :
======
Le buffer n'est pas gardé entre les appels de la fonction.
Pour l'implémenter soit :
    - je le passe en paramètre
    - soit je transforme la fonction en itérator
    - soit j'en fais une classe

La partie test est trop longue voir comment changer cela nose  etc...

"""

__version__ = "$Revision: 1232$"

#source $Source$


from numpy import *

def rif (datas,b_coefficients):
    """ Filtre RIF. Accepte un array de données et un array de coéficients et
    retourne un array de données sur laquelle a été appliquée l'opération de
```

```
    convolution soit datas * b_coefficients. L'array en sortie à la même
    dimension que celui passé en paramètre. Le buffer n'est pas consevé entre
    les appels.)
    """

    # initilialisation
    size_of_coefficient = len(b_coefficients)
    size_of_data = len(datas)

    x_buffer = zeros(size_of_coefficient)
    datas_ok = zeros(size_of_data)


    for element_data in xrange(size_of_data):
        # décaler le buffer = historique
        x_buffer[1:] = x_buffer[:-1]

        # ajouter la nouvelle valeur
        x_buffer[0] = datas[element_data]

        #calculer la réponse impulsionnelle

        y = x_buffer * b_coefficients
        datas_ok[element_data] = y.sum()

    return datas_ok


if __name__ == "__main__":
    ### vérification avec numpy
    from numpy.random import *
    from scipy.signal import *

    # initialisation

    number_tap = 6
    cut_frequency = 0.1

    # génération des coéfficients
    b_coefficients, a_coefficients = butter(number_tap, cut_frequency)
    # pour le test du filtre fir, il faut que a= 1.0
    a_coefficients=array([1.0])

    # test avec 100.000 valeurs
    test_datas = uniform(-1., 1., 100000)
    scipy_test_datas = lfilter(b_coefficients, a_coefficients, test_datas)
    rif_test_datas = rif(test_datas, b_coefficients)

    # le resultat doit rester faux pour être ok
    error = False
    for element in xrange(len(test_datas)):
        # attention il y a des problème d'arrondi

        if round(scipy_test_datas[element],12) != round(rif_test_datas[element],12):
            print (scipy_test_datas[element])
            print (rif_test_datas[element])
            error= True

    if error :
        print "le test vrai est NOK"
    else:
        print "tout est ok"
```

A.0.3 Les filtres à réponse impulsionnelle infinie

Voici en langage Python un filtre RII :

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

```
"""
Implémentation␣d'un␣filtre␣à␣réponse␣impulsitionnelle␣infinie.
Cette␣implémentaton␣se␣base␣sur␣le␣type␣array␣de␣numpy.

Pour␣verifier␣son␣implémentation,␣j'utilise␣signal. lfilter ␣de␣scipy.
De␣ce␣fait,␣j' utilise ␣la␣même␣notation␣pour␣les␣coffecients␣a␣et␣b.
Attention,␣à␣cause␣de␣problème␣d'arrondi,␣je␣n'utilise␣que␣12␣décimales
après␣la␣virgule.

TODO␣:
======
Le␣buffer␣n'est␣pas␣gardé␣entre␣les␣appels␣de␣la␣fonction.
Pour␣l'implémenter␣soit␣:
␣␣␣␣-␣je␣le␣passe␣en␣paramètre
␣␣␣␣-␣soit␣je␣transforme␣la␣fonction␣en␣itérator
␣␣␣␣-␣soit␣j'en␣fais␣une␣classe

La␣partie␣test␣est␣trop␣longue␣voir␣comment␣changer␣cela␣nose␣␣etc...

"""

__version__ = "$Revision:␣1232$"

#source $Source$


from numpy import *

def rii (datas,b_coefficients, a_coefficients):
    """␣Filtre␣RII.␣Accepte␣un␣array␣de␣données␣et␣deux␣array␣de␣coéficients␣et
␣␣␣␣retourne␣un␣array␣de␣données␣sur␣laquelle␣a␣été␣appliquée␣l'opération␣de
␣␣␣␣convolution.␣L'array␣en␣sortie␣à␣la␣même␣dimension␣que␣celui␣passé
␣␣␣␣en␣paramètre.
␣␣␣␣Le␣buffer␣n'est␣pas␣consevé␣entre␣les␣appels.
␣␣␣␣"""

    #   initilialisation

    size_of_coefficient = len(b_coefficients)
    size_of_data = len(datas)

    entree_buffer = zeros(size_of_coefficient)  # historique des entrees
    sortie_buffer = zeros(size_of_coefficient)  # historique des sorties
    data_ok = zeros(size_of_data)


    for element_data in xrange(size_of_data):
        # décaler les buffers = historique entree et sortie
        entree_buffer[1:] = entree_buffer[:-1]
        sortie_buffer [1:] = sortie_buffer[:-1]

        # ajouter la nouvelle valeur pour entree
        entree_buffer[0] = datas[element_data]

        #calculer la réponse impulsionnelle en deux temps

        sortie_buffer[0]= b_coefficients[0] * entree_buffer[0]

        sortie = (entree_buffer [1:] * b_coefficients [1:]) - \
                 (sortie_buffer [1:] * a_coefficients [1:])

        sortie_buffer [0] += sortie.sum()

        data_ok[element_data] = sortie_buffer[0]

    return data_ok


if __name__ == "__main__":
    ### vérification avec numpy
    from numpy.random import *
    from scipy.signal import *
```

```
# initialisation

number_tap = 6
cut_frequency = 0.1

# génération des coéfficients
b_coefficients, a_coefficients = butter(number_tap, cut_frequency)

# test avec 100.000 valeurs
test_datas = uniform(-1., 1., 100000)
scipy_test_datas = lfilter (b_coefficients, a_coefficients, test_datas)
rii_test_datas = rii (test_datas, b_coefficients, a_coefficients)

# le resultat doit rester faux pour être ok
error = False
for element in xrange(len(test_datas)):
    # attention il y a des problème d'arrondi
    test=round(scipy_test_datas[element],7) - round(rii_test_datas[element],7)

    if test != 0:
        print "*" * 40
        print "scipy ",(scipy_test_datas[element])
        print " rii    ", ( rii_test_datas [element])
        print "*" * 40
        error= True

if error :
    print "le test vrai est NOK"
else:
    print "tout est ok"
```

Annexe B

Les fichiers wav

B.1 En-tête d'un fichier wav

Voici d'après [Wikipedia, 2009i] et [Scott, 2003], l'en-tête d'un fichier wav.

```
[Bloc de déclaration du container RIFF]
   ChunkId          (4 octets) : Constante « RIFF »  (0x52,0x49,0x46,0x46)
   ChunkSize        (4 octets) : Taille du fichier moins 8 octets
   Format           (4 octets) : Format = « WAVE »  (0x57,0x41,0x56,0x45)

[Bloc décrivant le sous-container  fmt]
   SubChunk1Id      (4 octets) : Identifiant « fmt »  (0x66,0x6D, 0x74,0x20)
   SubChunk1Size    (4 octets) : Nombre d'octets du bloc - 8
   AudioFormat      (2 octets) : Format du stockage dans le fichier (1: PCM, ...)
   NumChannels      (2 octets) : Nombre de canaux
   SampleRate       (4 octets) : Frêquence d'échantillonnage (en Hertz)
   ByteRate         (4 octets) : Nombre d'octets à lire par seconde soit :
                                 SampleRate * NumChannels * BitsPerSample/8
   BlokAlign        (2 octets) : Nombre d'octets par bloc d'échantillonnage soit :
                                 NumChannels * BitsPerSample/8
   BitsPerSample    (2 octets) : Nombre de bits pour le codage (8, 16, 24)

[Bloc décrivant le sous-conatainer des données]
   SubChunk2Id      (4 octets) : Constante « data »  (0x64,0x61,0x74,0x61)
   SubChunk2Size    (4 octets) : Nombre d'octets des données soit :
                                 NumSpamples * NumChannels * BitsPerSample/8
   DATAS[] :        (..octets) : Ici commence les octets datas.
```

La taille de l'en-tête fait donc 44 octets. L'encodage est little endian sous un processeur Intel. Il y a des variantes big endian. Chaque échantillon est mis bout à bout. Les datas de chaque piste alternent successivement pour chaque échantillon comme suit : [Échantillons 1 du Canal 1] Échantillons 1 du Canal 2] [Échantillons 2 du Canal 1] [Échantillons 2 du Canal 2] etc... dans le cas de deux canaux.

B.2 Utilitaires d'exploration de fichiers wav

B.3 Génération du fichier exemple

Pour illustrer un fichier wav, nous avons généré un fichier audio dont les caractéristiques sont les suivantes :

- nombre de pistes : 2 ;
- durée : une seconde ;
- fréquence d'échantillonnage : 48 000 Hz ;
- encodage : 16 bits ;
- taille du fichier : 192 044 octets.

B.4 La commande shell ls

Nous pouvons vérifier grâce à la commande ls -al que le fichier à la taille requise.

```
$ ls -al foo.wav
-rw-r--r-- 1 gdh2 gdh2 192044 2009-04-14 17:23 foo.wav
```

Hexdump

Voici le header d'un fichier wav visualisé à l'aide de la commande *hexdump* [1] -n 44 foo.wav :

```
0000000 4952 4646 ee24 0002 4157 4556 6d66 2074
0000010 0010 0000 0001 0002 bb80 0000 ee00 0002
0000020 0004 0010 6164 6174 ee00 0002
```

Nous remarquons tout de suite qu'il s'agit d'un encodage little endian. En effet, EE00 0002 doit être interprété comme 2EE00 (soit 192.000). Ce qui correspond bien au nombre de bytes du fichier moins le header de 44 octets.

B.5 Sox

2

L'utilitaire *SoX* [3], véritable "*couteau suisse de la manipulation audio*" comme l'affirme la man page permet outre une série de transformations, de changements de format d'obtenir des statitistiques.

B.5.1 Play

Voici les statistiques obtenues avec play :

```
$play foo.wav

Input File     : 'foo.wav'
Sample Size    : 16-bit (2 bytes)
Sample Encoding: signed (2's complement)
Channels       : 2
Sample Rate    : 48000
```

[1] Présente dans les distribution Unix like : Ubuntu8.04.3
[2] Sound eXchange
[3] `http://sox.sourceforge.net`

B.5.2 Sox

Cet outil très polyvalent permet aussi d'obtenir les statistiques que voici :

```
sox foo.wav -e stat

Samples read:             96000
Length (seconds):      1.000000
Scaled by:         2147483647.0
Maximum amplitude:     0.999969
Minimum amplitude:    -1.000000
Midline amplitude:    -0.000015
Mean    norm:          0.496052
Mean    amplitude:     0.001235
RMS     amplitude:     0.574017
Maximum delta:         1.988281
Minimum delta:         0.000000
Mean    delta:         0.660569
RMS     delta:         0.810274
Rough   frequency:        10783
Volume adjustment:         1.000
```

B.6 Comparaison entre le flac et le wav

B.7 Comparaison entre le flac et le wav

B.7.1 Vitesse de décodage / d'encodage

Voici les résultats du test réalisé avec vitesse.py en local sur un processeur pentium m 1,6 Ghz qui tourne sur le système d'exploitation Ubuntu 8.04.2 :

fichier	roll		rock		foo		sin		stotzem		jazz	
Format	wav											
taille	238K	238K	595K	595K	563K	563K	33M	33M	44M	44M	70M	70M
type i/o	read	write	read	write	read	write	read	write	read	write	read	write
tps ms	281	288	273	448	300	275	439	769	479	883	564	2870
	286	276	304	334	320	303	438	737	459	845	575	1227
	306	265	272	289	270	340	417	722	468	895	575	1417
	298	261	309	292	263	296	474	769	489	902	560	1283
	282	309	303	301	316	325	427	730	479	878	571	1675
	313	253	298	290	288	297	450	788	448	888	586	2305
	285	296	316	325	278	300	443	734	490	844	577	1234
	294	279	310	276	304	302	445	751	502	866	605	1216
	273	296	317	342	288	304	441	739	449	849	560	1231
	304	293	288	291	267	308	406	707	453	874	608	1251
moyenne	292	282	299	319	289	305	438	745	472	872	578	1571
write/read	0,96		1,07		1,05		1,70		1,85		2,72	
Format	flac											
taille	162K	162K	433K	433K	564K	564K	9.5M	9.5M	25M	25M	38M	38M
type i/o	read	write	read	write	read	write	read	write	read	write	read	write
tps ms	351	360	323	300	321	344	1174	2964	1482	4971	2335	5762
	324	266	311	288	266	343	1200	2942	1522	3595	256	5588
	283	256	306	295	283	364	1181	2691	1511	3585	2305	5650
	307	326	344	338	299	347	1151	2623	1482	3613	2303	5628
	336	319	335	329	291	351	1189	2651	1516	3588	2281	5606
	288	314	300	295	287	326	1178	2642	1529	3971	2261	5691
	294	272	304	315	313	356	1164	2603	1507	3603	2303	5627
	314	261	316	323	293	332	1200	2654	1540	3601	2282	5622
	277	342	333	312	257	348	1183	2600	1503	3621	2258	5635
	301	350	331	315	292	360	1159	2663	1525	3605	2249	5668
moyenne	308	307	320	311	290	347	1178	2703	1512	3775	2083	5648
write/read	1,00		0,97		1,20		2,30		2,50		2,71	
flac/wav	1,05	1,09	1,07	0,98	1,00	1,14	2,69	3,63	3,21	4,33	3,60	3,60

Pour comparer la vitesse de décodage, nous avons utilisé deux programmes. Le premier a été réalisé avec le langage de scripting python. Pour le second, nous avons utilisé un programme compilé sox. Le script python charge simplement le fichier et le stocke dans un tableau. Il fait cela par chunk de 2048 échantillons. Notons que nous avons fait varier la taille du chunk sans noter de variation sensible. Nous avons donc choisi une taille de chunk qui n'alourdit pas la mémoire vive de l'ordinateur et qui renferme en même temps suffisamment d'informations lors d'un traitement éventuel.

Dans un premier temps, nous nous sommes limités à des opérations en local.

Comme on pouvait s'y attendre, le temps de chargement n'est pas proportionnel à la taille du fichier. Cela est dû au temps d'initialisation du périphérique qui est constant quelque soit la taille du fichier.

Les fichiers flac de moins de 600K sont aussi performants tant en écriture qu'en lecture. Il en va autrement dès que la taille du fichier augmente. Ici le temps processeur devient proportionnellement important puisqu'il faut décompresser les données.

Ces quelques résultats démontrent en tout cas qu'il ne faut pas tirer de conclusions sans tester effectivement son schéma de traitement. A la lueur de ces résultats, il semble que le format non compressé soit être privilégié. Cette vérification doit être faite dans le cas d'une distribution des tâches sur un cluster.

Vitesse d'encodage

Pour comparer la vitesse d'encodage, nous avons utilisé uniquement le même script python.

B.7.2 Test de vitesse de flac wav avec sox

Voici les résultats du test réalisé avec sox 14.0.0 au moyen de la commande sox nom_du_fichier -e stat en local sur un processeur pentium m 1,6 Ghz qui tourne sur un système d'exploitation Ubuntu 8.04.2 :

fichier	jazz.wav		jazz.flac
tps ms	3484		5864
	2973		5012
	3003		5027
	2976		5013
	2998		5029
	3022		4987
	3585		5163
	2973		4979
	2993		5020
	3652		4993
moyenne	3166		5109
flac/wav	1,61		

B.7.3 Le programme vitesse.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""
A documenter


"""

__version__ = "$Revision: 1232$"

#source $Source$


import numpy as np
from scikits.audiolab import Sndfile, Format
from optparse import OptionParser
##from sys import exit

class Sound(object):
    """Ce programme lit un fichier en entrée et le stocke dans un tableau.
    Il peut aussi l'écrire dans un fichier soit sous format wav soit
    sous format flac.
    Attention toutefois que le format wav est codé sur
    16 bits et que le format flac est lui compressé au niveau 5.
    Il faut considérer la conversion comme un effet de bord plutôt
    qu'une fonctionnalité.
    """

    # constructeur
    def __init__(self, fichierATraiter, fileFormat="wav", chunkSize=2048):
        self.__fichier_traite = fichierATraiter + "." + fileFormat
        self.__file_in = Sndfile( fichierATraiter, 'r')
        self.__nb_frames = self.__file_in.nframes
        self.__write = False
        self.__fs = self.__file_in.samplerate
        self.__nc = self.__file_in.channels
        self.__nb_frames_lues = 0
        self.__file_format = "wav"
        self.__nb_frames_lues = 0
        self.__chunk_size = chunkSize
        self.__data = []
        self.__file_out = "testcloug"
        if self.__chunk_size >= self.__nb_frames :
            self.__chunk_size = self.__nb_frames
        #nombre de chunk
        self.__chunk_nbr = int (float (self.__nb_frames)/ \
                                float (self.__chunk_size))


    def setFormat(self, fileFormat):
        """selectionne le format pour le fichier de sortie"""
        self.__file_format = Format(fileFormat)

    def setWrite(self, state):
        """permet ou non l'écriture sur disque"""
        self.__write = state
        if self.__write:
            self.setFormat(self.__file_format)
            self.__file_out = Sndfile(self.__fichier_traite, 'w', \
                self.__file_format, self.__nc, self.__fs)

    def iterator (self):
        """ lit le fichier et renvoie le nbr de frames définis par chunk_nbr"""

        for i in xrange(self.__chunk_nbr):
            # le dernier chunk

            if i == self.__chunk_nbr-1:
                self.__chunk_size = self.__nb_frames- self.__nb_frames_lues
            self.__data = self.__file_in.read_frames(self.__chunk_size)
```

```
            self.__nb_frames_lues += self.__chunk_size
            yield self.__data

    def writeTofile (self, data):
        """ ecrit les données"""
        if self.__write:
            self.__file_out.write_frames(data)


    def closeFile(self):
        """ferme le fichier ouvert en écriture"""
        if self.__write:
            self.__file_out.close()



def initParser ():
    """analyse la ligne de commande"""
    usage = "Usage: python Sound.py [options] filename"

    parser = OptionParser(usage)

    parser.add_option("--chunkSize", dest="chunkSize", type="int", \
                      default=2048, help="Size of chunk in frame")

    parser.add_option("--format", dest="fileFormat", type="string", \
                      default="wav", help="format of file" )

    parser.add_option("--write", dest="write", action="store_true" , \
                    default=False, help="write the file with extension ")


    (opt, args) = parser.parse_args()


    if len(args) != 1:
        print "Improper number of arguments.Please give one name's file"
        exit ()

    return (opt, args)


if __name__ == "__main__":
    ##(opt, args) = initParser ()
    ##print opt
    ##print args
    ###testSound = Sound(args[0], opt.fileFormat, opt.chunkSize)
    ##testSound.setWrite(opt.write)
    ##testSound.iterator()
    ##testSound.closeFile()
    test = Sound("stotzem_8.wav")
    gen = test. iterator ()
    test.setWrite(True)

    for compteur in gen:
        test. writeTofile (compteur)
    test.closeFile ()

    #pour tester il faudrait calculer une somme de controle par exemple.
```

B.7.4 Espace disque

Nom fichier	format wav	format flac	taux compression	description
rocky4	595K	339K	43 %	voix de femme
roll	238K	155K	35 %	son sinusoidal multifrequence
short_count	34K	8,0K	77 %	decompte voix d'homme
sin_440_30	33M	7,3M	77 %	son sinuosidal 440 Hz
train	28K	22	21 %	bruit d'un sifflet de train
bruit	563K	563K	0 %	bruit aléatoire
stotzem_8	44M	23M	48 %	guitare sèche

Remarquons que la performance du format flac fluctue fortement en fonction du contenu du fichier audio. Dans cet échantillon, il va de 0 % à 77 %. Son score le plus bas est obtenu avec un fichier audio qui contient du bruit généré de manière aléatoire. Son score le plus haut est obtenu avec un fichier qui contient un son sinusoïdal à 440 Hz. Notre analyse confirme bien que "plus le signal est constitué d'ondes régulières, meilleure est la compression". [Wikipedia, 2009d]

Annexe C

Image

C.1 Définitions

C.1.1 L'image

En nous basant sur la définition que donne l'Académie Française dans sa huitième édition [Académie Française,], une image est l'enregistrement sur un support des ondes émises, réfléchies ou filtrées par un objet ou une personne. Ces ondes peuvent être de toute nature : ultra-son, lumière visible, infrarouge, rayon X ou gamma par exemple. Cet enregistrement peut être soit continu, comme par exemple la réaction d'une émulsion photographique à la longueur d'onde (lumière visible ou ultraviolet), soit discret, ainsi en est-il du codage sous forme de valeur numérique. Le support physique pour les photos noir et blanc est constitué de grains d'argent et celui pour les photos couleur de flocons de colorant.

Une image peut être définie par sa largeur et sa hauteur, généralement exprimées en pixels. On cite d'abord la largeur (axe horizontal), puis la hauteur (axe vertical).

Il existe différentes définitions standardisées pour une image. Ces standards reflètent l'adoption de l'image numérique par le monde informatique d'abord et par le monde photographique (photo d'art et publicité) ensuite. Des formats tels que 640X480, 800X600, 1024X768 appartiennent plus au monde informatique. Un format tel que 3648x2736 pixels est en fait le format d'un panneau publicitaire de 4 mètre sur 3 mètres. Pour obtenir cette définition il faudra un capteur de 10Mpix avec un rapport d'image de 3/4.

La définition d'une image ne traduit pas directement la qualité de cette image.

C.1.2 La densité d'une image

Une autre notion est la densité de pixels par pouce. Elle traduit en fait la qualité de l'image. Cette notion va beaucoup intervenir car elle va introduire un bruit. (à développer)

C.1.3 La résolution d'une image

La **résolution** d'une image est sa traduction sur un support. Elle exprime le nombre de pixels de l'image occupée sur le support par pouce. Pour une définition de 300 x 900 pixels, l'image qui aura

une résolution de 300 pixels(pi) par pouce occupera donc une espace de 1 X 3 pouces sur le support.

C.1.4 La profondeur d'échantillonnage

Le nombre de bits utilisés par pixel et par couche est aussi appelé la **profondeur d'échantillonnage**. Une image de 24 bits rvb possède donc une profondeur de 8 bits puisque les 3 canaux (rvb) sont codés.

Actuellement, on trouve des capteurs avec une profondeur d'image de 12, 14 ou 16 bits. Une grande profondeur est toujours intéressante à condition que les algorithmes utilisés puissent en tirer parti. Ce qui n'est pas le cas de la compression JPEG. En effet, pour réaliser la compression, on utilise la DCT (Discrete Cosine Transform) qui n'est applicable comme nous le verrons plus loin qu'à une profondeur de 8 bits.

C.1.5 Les canaux d'une image

C'est le nombre de couches que contient verticalement une image. Les images binaires ou en niveaux de gris comportent un seul canal. Les images couleurs possèdent généralement trois à quatre canaux.

Précisons que le nombre de canaux d'une photo varie en fonction de l'usage que l'on désire en faire. On peut ainsi enregistrer dans une seule couche les informations infrarouges, ultraviolets et en lumière visible. Ce type d'image est surtout destiné à l'analyse ou à la recherche. Citons par exemple les images satellitaires de la Terre ou bien les clichés qui ont été pris avec différents filtres.

C.1.6 La transparence ou le canal alpha

Aujourd'hui, la transparence (nommée canal alpha) est parfois codée. C'est à travers ce pixel de l'image que "passera" en partie la couleur d'un pixel d'une autre image placée dans la même fenêtre, mais « derrière » la première image (technique dite de l'alpha blending en anglais).[Wikipedia, 2009a]. Dans notre travail, nous ne traiterons pas du canal alpha.

C.1.7 La qualité d'une image

Cette notion toute subjective fait pourtant intervenirun certain nombre de paramètres mesurables tels que : le type de bruit, l'histogramme ou les détails dans les hautes couleurs.

C.1.8 Le gamut

Contrairement à l'espace colorimétrique qui désigne un ensemble de couleurs que le système peut représenter, le gamut désigne l'enveloppe externe de cet espace colorimétrique.

C.1.9 Balance des blancs

La balance des blancs permet à l'appareil d'indiquer manuellement ou automatiquement le type de lumière qui est utilisé pour éclairer la scène photographiée. Lorsque cette opération est faite manuellement, on pointe l'objectif vers une source de couleur blanche. De manière théorique, la balance des blancs consiste donc à équilibrer les trois canaux RVB.

C.1.10 Gamma

Le gamma est la mesure du contraste d'une image (dans les niveaux de gris moyens).

C.2 Les méta-données

C.2.1 Introduction

En plus des informations du signal (intensité ou couleur), d'autres informations telles que la date de création, la date de modification, la distance focale, le profil colorimétrique ou, tout simplement, l'auteur, sont des données qui peuvent être stockées. Ces informations sont appelées méta-données.

Les méta-données permettent un traitement automatique des images. Les logiciels de traitement peuvent ajouter ou modifier les méta-données et ainsi apporter une plus-value à l'information initiale. Cette information peut être sémantique puisque des mots-clés peuvent êtres enregistrés.

Les méta-données sont importantes dans le milieu professionnel ainsi que lors du traitement de l'image.

C.2.2 Enregistrement des méta-données

Les méta-données se présentent sous trois formes :

- incorporées au fichier (elles cohabitent avec les données du signal) ;
- dans un fichier indépendant des données du signal ;
- stockées dans une base de données. Cette dernière solution permet un traitement plus rapide ainsi qu'une conservation plus efficace des informations.

C.2.3 Format des méta-données

Voici les principaux types de données. Les données IPTC, du nom de l'organisme International Press Telecomunication Coucil, sont des données qui permettent de décrire le contexte dans lequel le cliché a été pris. On peut utiliser les mots-clés par exemple. [IPTC, 2009] Les données EXIF sont des informations techniques que l'appareil enregistre lui-même. Il s'agit par exemple la durée d'exposition L'Extensible Metadata Platform ou XMP est un format de méta-données basé sur XML. Il a été lancé par Adobe Systems en avril 2001. [Wikipedia, 2009c].Bien qu'il soit ouvert à tout type de données pouvant intégrer un document XML, XMP prédéfinit la façon de stocker un certain nombre d'informations parmi les plus courantes en reprenant en particulier des éléments de Dublin Core et d'EXIF.

C.3 Les espaces colorimétriques en photo numérique

Si l'on travaille dans l'espace colorimétrique rvb, voici trois espaces courants dans le monde de la photo :

- **sRVB IEC61966-2.1** Sous ensemble de l'espace RVB, il est adapté au Web.
- **Adobe RVB (1998)** Gamut plus important que le précédent car il permet aussi la représentation sur papier.
- **ProPhoto RVB** Le plus riche des trois puisque son gamut englobe les deux précédents. Il permet aussi de traduire la sensibilité des appareils photo modernes puisqu'il permet un codage sur 16 bits.

Annexe D

Les listings

Typographical Conventions for Python

Python:Normal Text	**Python:Definition Keyword**	Python:Operator
Python:String Substitution	**Python:Command Keyword**	**Python:Flow Control Keyword**
Python:Builtin Function	Python:Special Variable	Python:Extensions
Python:Exceptions	**Python:Overloaders**	Python:Preprocessor
Python:String Char	Python:Long	Python:Float
Python:Int	Python:Hex	Python:Octal
Python:Complex	*Python:Comment*	Python:String
Python:Raw String	**Alerts:Normal Text**	**Alerts:Alert**

```
 1 #!/usr/bin/env python
 2 #-*- coding: utf-8 -*-
 3
 4 __author__ = "Charles Duysinx aka MonsieurCloug"
 5 __copyright__ = "Copyright 2009, Charles Duysinx"
 6 __license__ = "MIT"
 7 __version__ = "0.0"
 8 __revision__ = "$LastChangedRevision: 175 $"[22:-2]
 9 __date__ = "$LastChangedDate: 2009-03-17 16:15:55 +0100 \
10             (Mar, 17 Mar 2009) $"[18:-2]
11
12
13 import redlindalock.kernel as kernel
14 ##from redlindalock.kernel import connect, universe, uts, TupleSpace, \
15 ##      RedBlockError, RedLockTransactionError, RedLockTsError, \
16 ##      RedLockUseTsError, RedLockRedisError, lock, unLock, getLock
17
18
19 from redlindalock.kernel import *
20
21 # connect supression de connect qui surchargeait la  méthode de la classe Redis
22 ## même nom qu
23
```

Typographical Conventions for Python

Python:Normal Text	**Python:Definition Keyword**	Python:Operator
Python:String Substitution	**Python:Command Keyword**	**Python:Flow Control Keyword**
Python:Builtin Function	Python:Special Variable	Python:Extensions
Python:Exceptions	**Python:Overloaders**	Python:Preprocessor
Python:String Char	Python:Long	Python:Float
Python:Int	Python:Hex	Python:Octal
Python:Complex	*Python:Comment*	Python:String
Python:Raw String	Alerts:Normal Text	**Alerts:Alert**

```
 1 reDlindaHost = 'localhost'
 2 reDlindaPort = 6379
 3 reDlindaTimeOut= None
 4 reDlindaNodelay = None
 5 reDlindaCharset = 'utf8'
 6 reDlindaErrors = 'strict'
 7 reDlindaSock = None
 8 reDlindaFp = None
 9 reDlindaDb = 0
10
```

Typographical Conventions for Python

Python:Normal Text	**Python:Definition Keyword**	Python:Operator
Python:String Substitution	**Python:Command Keyword**	**Python:Flow Control Keyword**
Python:Builtin Function	Python:Special Variable	Python:Extensions
Python:Exceptions	**Python:Overloaders**	Python:Preprocessor
Python:String Char	Python:Long	Python:Float
Python:Int	Python:Hex	Python:Octal
Python:Complex	*Python:Comment*	Python:String
Python:Raw String	**Alerts:Normal Text**	**Alerts:Alert**

```
 1 #!/usr/bin/env python
 2 # -*- coding: utf-8 -*-
 3
 4
 5 __author__ = "Charles Duysinx aka MonsieurCloug"
 6 __copyright__ = "Copyright 2009, Charles Duysinx"
 7 __license__ = "MIT"
 8 __version__ = "0.0"
 9 __revision__ = "$LastChangedRevision: 175 $"[22:-2]
10 __date__ = "$LastChangedDate: 2009-03-17 16:15:55 +0100 \
11             (Mar, 17 Mar 2009) $"[18:-2]
12
13
14 import time
15 import sys
16 from utils import *  #une boites à outils pour ne pas polluer le module
17 from redis import *  #les primitives qui permmettent de communiquer avec Redis
18 from config import * #pour la configuration du serveur Redis
19
20
21 # création de ma classe d'erreur
22 class RedLindaError(Exception): pass
23
24 # non encore implémenté
25 class RedLindaErrorNotImplemented(RedLindaError): pass
26
27 # primitive blanquante non  exécutée
28 class RedBlockError(RedLindaError): pass
29
30 # impossible d'obtenir verrou de transaction
31 class RedLockTransactionError(RedLindaError): pass
32
33 # inconsistance au niveau de la gestion du verrou de transaction
34 class RedInconsistentLockTransactionError(RedLindaError): pass
35
36 # impossible de relacher le verrou de transaction
37 class RedrelaxeLockTransactionError(RedLindaError): pass
38
39
40 # impossilble de réaliser un copy ou un copy-collect
41 class RedLockUseTsError(RedLindaError): pass
42
43 # impossible d'obtenir un verrou global
44 # non encore implémenté
45 class RedLockRedisError(RedLindaError): pass
46
47 # impossible d'effacer la liste des opérations en cours
48 class RedListOpCurrentError(RedLindaError): pass
```

```
49
50
51
52
53 class TupleSpace(Redis):
54     """Cette classe implémente les primitives Linda en les traduisant pour le
55     serveur Redis.
56     """
57 ##     __slots__ = ( '_id','_sock' )
58
59 ##     __slots__ = ( '_fp','errors', 'logRequete','_sock', 'charset', \
60 ##     'db', 'delaiVerrouTransactionServeur', 'nodelay', 'host', 'timeout', \
61 ##      'delaiUnblockRequete', '_id', 'port', 'delaiBlockRequete')
62 ##
63
64     def __init__(self, tsId=None, gc=True):
65
66         #configuration Redis
67         self.setParamRedis()
68
69         #configuration Pylinda
70         print "voici l'id ", tsId
71         if tsId:
72             self._id = tsId
73         else:
74             self._id="0!"+ str(self.incr("redlinda:tuple:id"))
75             print "voici mon id", self._id
76
77         #configuration redLinda
78
79         # temps imparti pour obtenir le verrou de transaction
80         self.delaiVerrouTransactionServeur = 1000
81
82         # temps imparti pour éxécuter une transaction copy ou colletct
83         self.delaiTransactionTupleSpace = 1000
84
85         # temps imparti pour mettre un verrou pour global
86         # non implémenter à ce jour
87         self.delaiVerrouGlobalServeur = 1000
88
89
90         self.delaiUnblockRequete = 1000 # durée d'une requete non bloquante.
91         self.delaiBlockRequete = 1000   # durée d'une requete bloquante
92
93
94         self.gestionTransaction = 1000 #gestion des transactions c'est le type
95
96         self.logRequete = True              # demande de logger les demandes
97
98
99         self.listOpCurrent = []
100
101
102         self.connect()
103
104
105     def __getstate__ (self):
106
107         print "geststate"
```

```
108
109
110         #self.__slots__ à l'export ok
111
112
113         return {"_id":self._id,}
114
115
116     def __setstate__(self,dict):
117
118         self.__init__(dict['_id'])
119
120
121     def __getnewargs__(self,dict):
122         print "getnewargs"
123
124         return (dict['_id'])
125
126
127     def setHost(self,host):
128         self.host = host
129
130     def setPort(self,port):
131         self.port = port
132
133     def setTimeOut(self,timeOut):
134         self_timeout = timeOut
135         socket.setdefaulttimeout(self.timeout)
136
137     def setNoDelay(self,noDelay):
138         self.nodelay = noDelay
139
140     def setCharset(self,charset):
141         self.charset = charset
142
143     def  setErrors(self,errors):
144         self.errors = errors
145
146     def setSock(self,sock):
147         self._sock = sock
148
149     def selfFp(self,fp):
150         sefl._fp = fp
151
152     def setDb(self, db):
153         self.db = db
154
155     def getParamRedis(self):
156         listeParam=[]
157         listeParam.append(self.host)
158         listeParam.append(self.port)
159         listeParam.append(self.timeout)
160         listeParam.append(self.nodelay)
161         listeParam.append(self.charset)
162         listeParam.append(self.errors)
163         listeParam.append(self._sock)
164         listeParam.append(self._fp)
165         listeParam.append(self.db)
166         return listeParam
```

```
167
168
169     def setParamRedis(self):
170         self.host = reDlindaHost
171         self.port = reDlindaPort
172         self.timeout = reDlindaTimeOut
173         if self.timeout:
174             socket.setdefaulttimeout(self.timeout)
175         self.nodelay = reDlindaNodelay
176         self.charset = reDlindaCharset
177         self.errors = reDlindaErrors
178         self._sock = reDlindaSock
179         self._fp = reDlindaFp
180         self.db = reDlindaDb
181
182
183     def __del__(self):
184         """Destructeur de la classe. Nous ne faissons rien....
185
186         """
187
188     def _out(self, tup):
189         """écriture d'un tuple dans le tupleSpace.
190         Les tuples sont stockés sur le serveur redis sous formes de
191         clés=valeurs (keys=values).
192
193         Codage de la clé
194         ----------------
195         La clé est constituée d'un ensemble de valeurs séparées par deux
196         points.
197
198         Voici les valeurs qui seront présente dans la clé :
199         idPylinda = identifiant utilisé par Pylinda modifié
200         hash(tuple) = Ce hash est obtenu après avoir concataîné le type de
201                       chaque composant du tuple
202         len(tuple) = indique le nombre d'élements du tuple
203         data = une liste qui contient les valeurs de la clé
204
205         hasard+time = tentative de générer un nombre aléatoire
206                       je devrais utiliser une bibliothèque spécialisée.
207                       ou un système de hash.
208
209
210
211         Redis ne suppporte pas les les espaces dans les clés. C'est pourquoi
212         la clé sera transformée en utilisantl'"url encoding". L'"url encoding"
213         est bien supporté par les serveurs en général et par Redis en
214         Particulier.
215
216
217
218         Codage des datas
219         ----------------
220         datas= je sérialise le tuple tout simplement.
221
222
223         Choix du type de stockage sur Redis
224         -----------------------------------
225         Redis offre la possibilité de stocker des strings, des lists et des
```

```
226         ensemble.
227         Au départ le choix des listes semblait intéressant. Mais tant la
228         création que l'extraction n'est pas adpaté à notre implémentation.
229         Lors de l'écriture on ne peut écrire qu'un élement à la fois. Cela pose
230         problème puisque le tuple peut-etre récupérer avant la fin complète
231         de son écriture définitive sur le serveur Redis.
232         Si nous avions choisis cette solution, nous aurions été obligé de
233         mettre un lock lors de l'écriture et de l'enlever lorsque tous les
234         élements du tuple aurait été stockés sur le serveur Redis.
235         La récupération d'une liste se fait en trois étapes si on ne connait
236         par la clé de la liste, ce qui est notre cas.
237         La première étape récupérer la clé, la seconde la longueur de la liste
238         et enfin la troisième récupère le contenu de la liste.
239         Vu ces désavantage, nous avons préférer utiliser le stockage de nos
240         tuples sous formes de string. Mais comme nous voulons garder les types
241         de nos élements de nos tuples nous créons d'abords une liste des
242         élements que nous sérialisons et puis nous sérialisons la lise.
243         Pour la sérialisation nous utilisons la sérialisation sous forme de
244         string.
245         Par mesure de sécurité cette chaines sera elle-même encodées en
246         utilisant l'"url encoding".
247
248         Malgrè la lourdeur de ce traitement les avantages sont intéressant:
249         tous les types de python qui sont sérialisables sont utilisables.
250         La richesses des types offerte par Python est donc conservées.
251
252         """
253
254         if type(tup) is not tuple:
255             raise TypeError, "Ceci %s n'est pas un tuple mais bien" %(type(tup))
256
257         msgTs=codeTs(self._id,tup) #msgTs sera un liste avec [keys, values]
258         print "voici la keys", msgTs[0]
259         print "voici les data", msgTs[1]
260
261         self.set(msgTs[0],msgTs[1])
262
263     def _rd(self, template):
264         """lecture blocante sans destuction d'un tuple dans le tupleSpace
265
266         """
267         if type(template) is not tuple:
268             raise TypeError, "Ceci %s n'est pas un tuple  " % (type(template))
269
270         #inititialisation
271
272         # impression de la requète
273         print "La demande était ", template
274
275         # initialisation des variables membres pour la directive Linda
276
277         self.unblock = False
278         self.preserveTuple = True
279
280         #traitement de la requete
281
282         reponseTuple =self.requeteUnique(template)
283
284         return reponseTuple
```

```
285
286
287
288     def analyseRequeteTs(self,ts):
289         # attention la présence du verrou globale doit-être vérifiée avant
290
291         if self.gestionTransaction == 1:
292             print "gestion transaction verrouilage ",  self.gestionTransaction
293             # par implémenation c'est une réponse constante
294             # je suis le seul à bosser"
295             # je garde le verrou transactionnel et j'exécute
296             result=[0,1]
297
298             return result
299
300
301         elif self.gestionTransaction == 2:
302             """pour le traitement j'ai à ma disposition
303             self._id = le tuple d'origine
304             ts  le tuple de destination
305             self.signatureTuple
306             self.preserveTuple pour savoir si c'est copy ou copy-collect
307
308             """
309
310
311             ## attention relacher le verrou de transaction
312             self.deVerrouillageTransaction()
313             print "gestion transaction verrouillage ",  self.gestionTransaction
314             raise RedLindaErrorNotImplemented,  self.gestionTransaction
315
316         elif self.gestionTransaction == 3:
317             ## attention relacher le verrou de transaction
318             self.deVerrouillageTransaction()
319             print "gestion transaction verrouilage ",  self.gestionTransaction
320             raise RedLindaErrorNotImplemented, self.gestionTransaction
321
322
323     def analyseRequeteTu(self,  signatureTuple):
324         # attention la présence du verrou globale doit-être vérifiée avant
325
326         if self.gestionTransaction == 1:
327             print "gestion transaction verrouilage ",  self.gestionTransaction
328             # par implémenation c'est une réponse constante
329             analyse = [0, 1]
330             return analyse
331
332         elif self.gestionTransaction == 2:
333             ## attention relacher le verrou de transaction
334             self.deVerrouillageTransaction()
335             print "gestion transaction verrouillage ",  self.gestionTransaction
336             raise RedLindaErrorNotImplemented,  self.gestionTransaction
337
338         elif self.gestionTransaction == 3:
339             ## attention relacher le verrou de transaction
340             self.deVerrouillageTransaction()
341             print "gestion transaction verrouilage ",  self.gestionTransaction
342             raise RedLindaErrorNotImplemented, self.gestionTransaction
343
```

```
344     def deVerrouillageOp(self):
345         """utilise la variable membre listOpCurrent et supprime toutes
346         les occurences sur le serveur redis
347 
348         """
349 
350         for element in self.listOpCurrent:
351             result=self.delete(element)
352             if not result:
353                 # on libère le verrou transactionnel
354                 self.deVerrouillageTransaction
355 
356                 raise RedListOpCurrentError,\
357                 " impossible de supprimer un élement en cours "+ str(element)
358 
359 
360 
361     def requeteGlobale(self,ts,template):
362 
363         """Pour la copie il me faut les id des tuples d'origine et de
364         destination.
365         Nous partons du principe que le programmeur passe réellement des
366         tuples qui sont effectivement définis, sinon on arrête le programme en
367         signalant l'erreur.
368 
369         Les id des tupleSpaces sont fournis pour :
370             le tupleSpace d'origine par self._id
371             le tupleSpace de destination par ts.self._id
372 
373         dans un soucis, de rapidité, j'ai choisi que le client fasse les
374         opération gourmande en temps avant de prendre contact avec le serveur
375         quitte à ce que cela soit superflu.
376 
377         """
378 
379         #codage pour la requete
380         requete = codeTemplate(self._id, template)
381 
382         # codage pour le lock ?
383         # je ne dois code que pour le tuple
384 
385         self.signatureTuple=signatureTuple(template)
386 
387 
388 
389         # j'enregistre ma demande sur le serveur si nécessaire
390         if self.logRequete:
391             logRequeteWrite()
392 
393         reponseValide = False
394         tpsAttenteReponse = 0
395         delaiReponse = 0
396         incrDelaiReponse = 0
397 
398 
399         attentePourExecution = self.acquisitionTransaction()
400 
401         # je vérifie que je n'ai pas de verrou global
402 
```

```
403         verrouGlobal=getLock()
404         if str(verrouGlobal)==str(redLindaAgentId)or verrouGlobal==None:
405             print "c'est bien moi",verrouGlobal
406             executionRequete=True
407 
408         else:
409             print "c'est pas moi",verrouGlobal
410             # je libère le jeton transaction
411             self.deVerrouillageTransaction()
412 
413 
414             attentePourExecution=False
415 
416 
417         # relaxe du verrou de transaction
418         if executionRequete == False:
419             self.deVerrouillageTransaction()
420 
421         # initialisation des compteurs
422         tpsTransactionTupleSpace = 0
423         incrDelaiVerrou = 0
424         delaiVerrou = 0
425         while attentePourExecution:
426             """je ne n'interresse qu'aux tuples qui ont la signature que
427             je veux déplacer et pas aux autres ce qui correspond à trois
428             premier champs de la requete:
429             lock:L:idTuple:signatureTuple:longueurTuple: pour un tuple
430             lock:G:R pour le lock global sur Redis
431 
432             """
433             # j'obtiens en retour une liste [0 0]
434             # ici comme l'opératin n'est pas bloquante je peux
435             # avoir un [0 ,0] donc je garde et je n'éxécute pas
436 
437             self.resultatAnalyseRequete=self.analyseRequeteTs(ts)
438             print "resulat analyse requete", self.resultatAnalyseRequete
439             print "resulat analyse requete", self.resultatAnalyseRequete[1]
440 
441 
442             if self.resultatAnalyseRequete[1]:
443 
444                 # je peux exécuter donc ...
445                 attentePourExecution = False
446 
447             # relaxe du verrou de transaction
448             if self.resultatAnalyseRequete[0]:
449                 self.deVerrouillageTransaction()
450 
451 
452             #Les trois premier champs de requeste
453             if not self.resultatAnalyseRequete[1]:
454                 # je ne peux pas exécuter la requete
455 
456                 if tpsTransactionTupleSpace <= self.delaiTransactionTupleSpace:
457 
458 
459                     tpsAttenteVerrouTransaction , delaiVerrou,incrDelaiVerrou =  \
460                     temporisateur(tpsAttenteVerrouTransaction,delaiVerrou, \
461                                                         incrDelaiVerrou)
```

```
462
463                 else:
464                     # pour que le système sorte dans un état stable
465                     self.deVerrouillageTransaction()
466                     raise RedLockUseTsError , \
467                         "Lock obtenu mais non utilisable sur le Ts"
468             else:
469
470                 attentePourExecution = False # et donc je peux passer à l'éxécution
471
472
473         if self.resultatAnalyseRequete[0]:
474             self.deVerrouillageTransaction()
475
476         # à ce stade je dois toujours exécuter la transaction

477         #envoi de la requete template
478         reponseTemplate=self.keys(requete)
479         print "voilà la requete", requete
480         if len (reponseTemplate) :
481             nbTuplesTraite = len(reponseTemplate)
482             # si j'ai un contenu je le récupère
483
484             if self.preserveTuple:
485                 """rename efface évidement la source on aurait
486                 aimé avoir un copy mais bon....
487                 """
488                 for tuple in reponseTemplate :
489                     newTuple = createTupleForCopy(ts._id,tuple)
490                     # on récupérue le tuple sur le serveur
491                     dataTuple=self.get(tuple)
492                     # on renome la clé du tuple
493                     reponseServeur = self.rename(tuple,newTuple)
494                     # on recree le tuple sur le serveur
495                     reponseServeur = self.set(tuple, dataTuple)
496
497                 self.deVerrouillageOp()
498                 if  not self.resultatAnalyseRequete[0]:
499                     self.deVerrouillageTransaction()
500
501             else:
502                 for tuple in reponseTemplate :
503                     newTuple = createTupleForCopy(ts._id,tuple)
504                     reponseServeur = self.rename(tuple,newTuple)
505
506                 self.deVerrouillageOp()
507                 if  not self.resultatAnalyseRequete[0]:
508                     self.deVerrouillageTransaction()
509
510         else :
511             self.deVerrouillageOp()
512             if  not self.resultatAnalyseRequete[0]:
513                     self.deVerrouillageTransaction()
514
515             nbTuplesTraite = 0
516
517         #efface ma demande sur le serveur
518         if self.logRequete:
519             logRequeteDelete()
```

```
520
521         return nbTuplesTraite
522
523     def requeteUnique(self,template):
524         # attention je reçois bien un liste... qui contient des tuples
525
526         reponseTuple=[]
527         nonReponseValide = False
528
529         #codage pour la requete
530         requete = codeTemplate(self._id, template)
531
532         #codage pour le lock
533
534
535
536         # j'enregistre ma demande sur le serveur si nécessaire
537         if self.logRequete:
538             logRequeteWrite()
539
540         reponseValide = False
541         tpsAttenteReponse = 0
542         delaiReponse = 0
543         incrDelaiReponse = 0
544         while not reponseValide:
545             self.acquisitionTransaction()
546
547             # dans tous les cas j'ai le verrou car acquisitionTransaction génére une
548             # une exception dans le cas contraire
549
550             # je vérifiie s'il y un lock global qui me bloque
551             # je suis imnunisé contre mes propres locks globaux
552
553             verrouGlobal=getLock()
554             if str(verrouGlobal)==str(redLindaAgentId)or verrouGlobal==None:
555                 print "c'est bien moi",verrouGlobal
556                 # j'analyse la transaction
557                 # je fabrique la signature du tuple
558
559                 self.signatureTuple=signatureTuple(template)
560
561
562                 """ C'est analyse requete qu'il faut implémenter la gestion des
563                     transactions, elle agit en fonction de la variable membre
564                     qui est self.gestionTransaction analyse requete retourne
565                     un liste avec deux infos :
566                     peut-on relacher le verrou transactionnel [0]
567                     peut-on exécuter la requète [1]
568
569                     On ne peut jamais avoir en retour l'info [0,0]
570
571                 """
572                 self.resultatAnalyseRequete=self.analyseRequeteTu(self.signatureTuple
    )
573
574             else:
575                 print "c'est pas moi",verrouGlobal
576                 # je libère le jeton transaction
577                 self.resultatAnalyseRequete= [1,0]
```

```
578
579
580                 self.resultatAnalyseRequete=self.analyseRequeteTu(self.signatureTuple
    )
581
582             if  (not self.resultatAnalyseRequete[0] ) and (not
    self.resultatAnalyseRequete[1] ):
583                 #situation où on ne peut pas exécuter la requet et que l'on doit
    garder le verrou de transaction
584
585                 self.deVerrouillageTransaction()
586                 self.deVerrouillageOp()
587
588
589                 raise RedInconsistentLockTransactionError,  \
590                     "Inconsistence au niveau des locks des transactions"
591
592             # relaxe du verrou de transaction
593             if self.resultatAnalyseRequete[0]:
594                 self.deVerrouillageTransaction()
595
596             #envoi de la requete template si autorisation
597             if  self.resultatAnalyseRequete[1]:
598                 print "voici la requete pour le serveur ", requete
599                 reponseTemplate=self.keysOne(requete)
600                 print "voilà la réponse", reponseTemplate
601
602
603                 if len (reponseTemplate) :
604                     # si j'ai un contenu je le récupère
605                     reponseTuple = self.get(reponseTemplate[0])
606
607
608                     if  not self.preserveTuple:
609                         reponseServeur = self.delete(reponseTemplate[0])
610                     else :
611                         reponseServeur = self.delete(reponseTemplate[0])
612                         # je le réecris sur le serveur
613
614                         keyWithNewTimeId = changeTimeId(reponseTemplate[0])
615                         supercloug=self.set(keyWithNewTimeId,reponseTuple)
616
617                     # j'efface ma demande sur le serveur
618
619                     if self.logRequete:
620                         logRequeteDelete()
621                     reponseValide = True
622                     self.deVerrouillageOp()
623                     self.deVerrouillageTransaction()
624
625
626                 else:
627                     self.deVerrouillageOp()
628                     self.deVerrouillageTransaction()
629
630                     if self.unblock:
631                         if tpsAttenteReponse <= self.delaiUnblockRequete:
632
633                             tpsAttenteReponse , delaiReponse,incrDelaiReponse =  \
```

```
634                             temporisateur(tpsAttenteReponse,delaiReponse,\
635                                                             incrDelaiReponse)
636
637                         else:
638                             reponseTuple=()
639                             reponseValide = True
640                     else:
641                         if tpsAttenteReponse <= self.delaiBlockRequete:
642                             tpsAttenteReponse , delaiReponse,incrDelaiReponse =  \
643                                     temporisateur(tpsAttenteReponse,delaiReponse,\
644                                                                 incrDelaiReponse)

645
646                         else:
647                             raise RedBlockError, \
648                                "Dépassement temps pour une directive bloquante"
649
650
651         # a effacer après
652         print "la réponse codée a été ", reponseTuple
653         print "La réponse a été       ", deCodeTs(reponseTuple)
654
655         #désérialisation et transformation d'un chaine unicode en string
656         #via deCodeTs
657
658
659         return deCodeTs(reponseTuple)
660
661
662     def _in(self, template):
663         """lecture blocante avec destuction d'un tuple dans le tupleSpace
664
665         """
666         if type(template) is not tuple:
667             raise TypeError,  "Ceci %s n'est pas un tuple  " % (type(template))
668
669         # impression de la requète
670         print "La demande était ", template
671
672         # initialisation des variables membres pour la directive Linda
673
674         self.unblock = False
675         self.preserveTuple = False
676
677         #traitement de la requete
678
679         reponseTuple =self.requeteUnique(template)
680
681         return reponseTuple
682
683
684     def _rdp(self, template):
685         """lecture non blocante sans destuction d'un tuple dans le tupleSpace
686
687         Notre implémentation, ne se se base pas sur la sémantique proposée par
688         Jacob and Wood contrairement à Pylinda.
689         [todo: décrire la proposotion]
690
691
```

```
692         """
693         if type(template) is not tuple:
694             TypeError,  "Ceci %s n'est pas un tuple  " % (type(template))
695 
696         # impression de la requète
697         print "La demande était ", template
698 
699         # initialisation des variables membres pour la directive Linda
700 
701         self.unblock = True
702         self.preserveTuple = True
703 
704         #traitement de la requete
705 
706         reponseTuple =self.requeteUnique(template)
707 
708         return reponseTuple
709 
710     def _inp(self, template):
711         """lecture non blocante avec destuction d'un tuple dans le tupleSpace
712 
713         Notre implémentation, ne se se base pas sur la sémantique proposée par
714         Jacob and Wood contrairement à Pylinda.
715         [todo: décrire la proposotion]
716         """
717 
718         if type(template) is not tuple:
719             raise TypeError, "Ceci %s n'est pas un tuple  " % (type(template))
720 
721         # impression de la requète
722         print "La demande était ", template
723 
724         # initialisation des variables membres pour la directive Linda
725 
726         self.unblock = True
727         self.preserveTuple = False
728 
729         #traitement de la requete
730 
731         reponseTuple =self.requeteUnique(template)
732 
733         return reponseTuple
734 
735     def collect(self, ts, template):
736         """Déplacement avec destuction dans le tuple d'origine vers
737         un nouveau tupleSpace.
738         Par soucis de compatiblité je dois retourner le nombre de tuples
739         déplacé.
740         En effet, on s'attend à ce que le programmeur exécute une lecture
741         destructive (donc un in).
742         On pourrait lui demander d'éxécuter une opération non blocante mais
743         A ce moment là la compatibilité est fortement remise en question.
744 
745 
746 
747         """
748         if ts.__class__ != TupleSpace:
749             raise TypeError, "Cet objet : %s doit être un tupleSpace" \
750                                 % (ts.__class__)
```

```
751         if type(template) is not tuple:
752             raise TypeError, "Ceci %s n'est pas un tuple  " % (type(template))
753 
754         self.preserveTuple = False
755         nbElemMove = self.requeteGlobale(ts,template)
756 
757         return nbElemMove
758 
759     def copy_collect(self, ts, template):
760         """Déplacement avec sans dans le tuple d'origine vers
761         un nouveau tupleSpace.
762         Par soucis de compatiblité je dois retourner le nombre de tuples
763         déplacés.
764         En effet, on s'attend à ce que l'utilisateur exécute une lecture
765         destructive.(donc un in)
766         On pourrait lui demander d'éxécuter une opération non blocante mais
767         A ce moment là la compatibilité est fortement remise en question.
768         Donc on interdit le changement sur les tuples.(pour éviter qu'il y ait
769         consomation, on vérifie si le tupleSpace de
770 
771 
772         """
773         if ts.__class__ != TupleSpace:
774             raise TypeError, "Cet objet : %s n'est pas un tupleSpace" \
775                                 % (ts.__class__)
776         if type(template) is not tuple:
777             raise TypeError, "Ceci %s n'est pas un tuple  " % (type(template))
778 
779         self.preserveTuple = True
780         nbElementCopy = self.requeteGlobale(ts,template)
781 
782         return nbElementCopy
783 
784 
785     def verrouillageTransaction(self):
786         result=self.set("red:lock:tr", str(redLindaAgentId), preserve=True)
787 
788 
789         return  result
790 
791 
792     def deVerrouillageTransaction(self):
793 
794         result =self.delete("red:lock:tr")
795         if result ==0:
796             raise RedrelaxeLockTransactionError, \
797                 "imposssible de relacher le jeton de transaction"
798 
799         return True
800 
801 
802     def acquisitionTransaction(self):
803 
804         verrouTransaction = False
805         tpsAttenteVerrouTransaction = 0
806         incrDelaiVerrou = 0
807         delaiVerrou = 0
808 
809         # cycle pour le lock de transaction
```

```
810         while not verrouTransaction:
811             verrouTransaction = self.verrouillageTransaction()
812 
813             if not verrouTransaction:
814                 if tpsAttenteVerrouTransaction <=
    self.delaiVerrouTransactionServeur:
815 
816                     tpsAttenteVerrouTransaction , delaiVerrou,incrDelaiVerrou =  \
817                     temporisateur(tpsAttenteVerrouTransaction,delaiVerrou, \
818                                                         incrDelaiVerrou)
819 
820                 else:
821                     raise RedLockTransactionError , \
822                         "Impossible d'obtenir un verrou de transaction"
823         return True
824 
825 
826     def __str__(self):
827         """represéentattion de la
828         """
829         if self._id == "0!0":
830             return "<Universal Tuplespace>"
831         else:
832             return "<TupleSpace %s >" % (self._id)
833 
834 
835     def __repr__(self):
836         """\brief Get a string representation of the tuplespace
837         """
838         if self._id == "0!0":
839             return "<Universal Tuplespace>"
840         else:
841             return "<TupleSpace %s >" % (self._id)
842 
843     __safe_for_unpickling__ = True
844 
845 
846 
847 def lock():
848 
849     """permet de protéger les sections critiques
850     met le verrou général
851     Pour rappel lock ont la syntaxe suivanate :
852     redLindaAgentId:lock:[ge:|ts|tu]:[dell|read|writ]:hash(signature):len(tuple)
853 
854      Attention l'implémentation de setnx est en fait la suivante
855     def set(self, name, value, preserve=False, getset=False):
856     Attention nous ne gérons le délai d'attente pour la mise en place du verrou
857     c'est au programmeur à le faire.
858 
859     """
860     result=universe.set("red:lock:gl", str(redLindaAgentId), preserve=True)
861 
862     return result
863 
864 def unLock():
865     """permet de protéger les sections critiques"
866     enlève le verrou général
867 
```

```
868     """
869     result=universe.delete("red:lock:gl")
870
871     return result
872
873
874 def getLock():
875
876
877     return universe.get("red:lock:gl")
878
879
880
881 def connect(cport=6379):
882
883     return True
884
885     #je vérifie et je retourne
886
887
888 ## création du tupleSpace Universe
889
890 universe = TupleSpace("0!0", False)
891
892 #### attention variable gobale n'est pas threadsave
893 ##creation de l'id agent
894 redLindaAgentId = universe.incr("red:agent:id")
895 print "voici l'id du client", redLindaAgentId
896
897
898 # pour des raisons de compatibilité mais à vérifier
899 uts = universe
900
901
902 if __name__ == '__main__':
903     print "je suis un module"
904
905
906
```

Typographical Conventions for Python

Python:Normal Text	**Python:Definition Keyword**	Python:Operator
Python:String Substitution	**Python:Command Keyword**	**Python:Flow Control Keyword**
Python:Builtin Function	Python:Special Variable	Python:Extensions
Python:Exceptions	**Python:Overloaders**	Python:Preprocessor
Python:String Char	Python:Long	Python:Float
Python:Int	Python:Hex	Python:Octal
Python:Complex	*Python:Comment*	Python:String
Python:Raw String	**Alerts:Normal Text**	**Alerts:Alert**

```
# -*- coding: utf-8 -*-

""" Ce fichier contient des choses bien utiles comme son nom l'indique

"""
# TODO: réorganiser les fonctions par thèmes

import urllib;
import thread
import time
import random
import zlib
from cPickle import loads, dumps
import redis as red


__author__ = "Charles Duysinx aka MonsieurCloug"
__copyright__ = "Copyright 2009, Charles Duysinx"
__license__ = "MIT"
__version__ = "0.0"
__revision__ = "$LastChangedRevision: 175 $"[22:-2]
__date__ = "$LastChangedDate: 2009-03-17 16:15:55 +0100 \
           (Mar, 17 Mar 2009) $"[18:-2]
zip = 1


def codeTs(id, tup):
    #la clé
    typeTup = ""
    dataCle = ""

    for elem in tup:
        typeTup = typeTup + str(type(elem))
        #dataCle = dataCle + urllib.quote(str(elem) )+ ":"
        dataCle = dataCle + str(hash(str(elem))) + ":"

    print "voici le typeTup",typeTup

    #hashTypeTup=str(hash(typeTup))
    #lenSignature=str(len(tup))

    #non appel à une procédure pour accélérer le traitement

    timeId = str(random.randint(1,10000000)) + str(time.time())

    """#cle = pyId:hash de la signature: longueur de la signature:
    url encoding de chaque data séparées par :
                                    identifiant de tuple unique)
```

```
49
50      """
51      cle = id+ ":" + str(hash(typeTup)) + ":" + str(len(tup)) + \
52                                                  ":" + dataCle + timeId
53
54
55      #les datas
56      datas=[]
57
58  ##      for elem in tup:
59  ##          datas.append(dumps(elem))
60  ##
61  ##      datasSerial=urllib.quote(dumps(datas))+"REDLINDA"+cle
62
63      # je sérialise le tuple... et non chaque élement du tuple
64      # je compresse le tuple
65
66      if zip == 1 :
67          datasSerial=urllib.quote(zlib.compress(dumps(tup), 9))
68      else:
69          datasSerial=urllib.quote(dumps(tup))
70
71
72      return [cle ,datasSerial]
73
74
75  def changeTimeId(keyString):
76      timeId = str(random.randint(1,10000000)) + str(time.time())
77      position =keyString.rfind(":") + 1
78      keyString=keyString[:position]
79
80      keyString=keyString + timeId
81
82      return keyString
83
84
85  def codeLockTuple(requete):
86      pass
87      return
88
89
90  def createTupleForCopy(id,tuple):
91      #attention le tuple n'est pas modif
92
93      listeTuple = tuple.partition(":")
94
95      print "debut",listeTuple
96      tupleRetour=[]
97
98      for element in listeTuple:
99          tupleRetour.append(element)
100
101     tupleRetour[0] = id
102     tupleRetour = "".join(tupleRetour)
103     print "tupleReou", tupleRetour
104     return tupleRetour
105
106
107 def codeTemplate(id, tup):
```

```
108
109     """ pour récupérer le tuple, j'ai besoin de deux type d'informations celle
110     qui sont implicites à mon implementation :
111     - de l'pyId du tuple,
112     - du hash de la signature du tuple,
113     - de la longueur,
114
115
116     et celle que le programmeur formule dans sa requete.
117
118     Remarquons que timeId n'est pas utilisé car il sert en quelque sorte de
119     hash.
120
121     Remarquons que les instances sont simplement codées dans la clé
122     avec le mot clé instance. En effet, les seuls instances qui nous
123     inressent ici sont objet tupleSpace. Il serait possible de stocker
124     d'autres types d'objet avec le serveur Redis. Il faudrait s'assurer
125     alors d'obtenir un identificateur unique auprès du serveur Redis.
126     Une autre manière de faire est de récupérer une instance et d'utiliser
127     l'introspcetion pour s'assurer que l'objet est bien celui que l'on attend.
128     Cette dernière solution est quand même très hasardeuse.
129
130     Attention, les instances tupleSpaces, sont "printable" car elles
131     possèdent un attribut __str__. Mais je n'utilise pas cette fonctionnalité
132     ici.
133
134
135     Remarquons aussi que les types des templates sont toujours évalué quant
136     au type après évaluation de l'expression.
137     Donc a=["un",bool,3,rouge+1] est bien évalué comme suit :
138     "un"     -> str           : string
139     bool     -> <type bool>   : un type booleen
140     3        ->int            : un entier
141     rouge+1 ->int             : si rouge est int alors la somme est int. Notre
142     implémentation ne considère pas qu'il s'agit d'un type somme.
143
144     Notre implémentation différencie les <type de ...> et les types. Quant on
145     désigne directement un type, il est remplacé par star dans la recherche.
146
147     """
148
149     typeTup=""
150     dataCleStar=""
151     cle=""
152
153     for elem in tup:
154
155
156         # <type 'instance'> est renvoyé pour une classe
157
158         if str(type(elem))=="<type 'instance'>":
159             typeTup = typeTup + str(type(elem))
160             #dataCleStar = dataCleStar + urllib.quote(str(type(elem)) )+":"
161             dataCleStar = dataCleStar + str(hash(str(type(elem))) )+":"
162
163         elif str(type(elem))=="<type 'type'>":
164             typeTup = typeTup + str(elem)
165             dataCleStar = dataCleStar+"*"+":"
166
```

```
167         else:
168             typeTup = typeTup + str(type(elem))
169             #dataCleStar = dataCleStar + urllib.quote(str(elem)) +":"
170             dataCleStar = dataCleStar + str(hash(str(elem))) +":"
171 
172 
173     cle = id+ ":" + str(hash(typeTup)) + ":" + str(len(tup)) + \
174                                                     ":" + dataCleStar + "*"
175 
176 
177     return cle
178 
179 def signatureTuple(tup):
180 
181     typeTup=""
182     signature=""
183 
184     for elem in tup:
185 
186         # <type 'instance'> est renvoyé pour une classe
187         # attention c'est u choix d'implémentation que ne tenir que du type
    d'instance
188 
189         if str(type(elem))=="<type 'instance'>":
190             typeTup = typeTup + str(type(elem))
191 
192         elif str(type(elem))=="<type 'type'>":
193             typeTup = typeTup + str(elem)
194 
195         else:
196             typeTup = typeTup + str(type(elem))
197 
198 
199     return str(hash(typeTup))
200 
201 
202 
203 
204 def deSerial(serialTup):
205     return loads(serialTup)
206 
207 
208 
209 def deCodeTs(ts):
210     """Attention on joue ici avec les strings et les unicodes.
211     Cela est vraiment très dangereux.... Vivement python 3
212     qui sera complètement unicode.
213 
214     """
215     tupleString=ts
216     #il peut recevoir un tuple vide
217 
218     if len(ts):
219 
220         if zip == 1 :
221             tsDezip= urllib.unquote(ts.encode())
222             tsDezip=zlib.decompress(tsDezip)
223             tupleString=loads(tsDezip)
224         else :
```

```
225             tupleString=loads(urllib.unquote(ts.encode()))
226
227
228     return tupleString
229
230
231 def codeUrl(ts):
232     """Encode chaque élément du tuple.
233     Ceci est utilisé car Redis ne supporte pas les espaces et autres caractères
234     dans les keys.
235
236     J'utilise ici l'url encoding" qui propose une méthodes reconnue par la
237     majorité des navigateurs webs et des serveurs.
238
239     """
240
241     return urllib.quote(ts);
242
243
244 def deCodeUrl(ts):
245     """Encode chaque élémement du tuple.
246     Ceci est utilisé car Redis ne supporte pas les espaces et autres caractères
247     dans les keys.
248
249         J'utilise ici l'url encoding" qui propose une méthodes reconnue par la
250     majorité des navigateurs webs et des serveurs.
251
252     """
253
254     return urllib.dequote(ts);
255
256 def createTimeId():
257     """Pour différencier les tuples entre eux qui sont produit par un même
258     client, j'uilise la concaténation du temps machine et d'un nombre au
259     hasard. Attention c'est un pseudo hasard.
260
261     Je me demande si je ne devrais pas simplement demander un nombre
262     unique au serveur redis.
263
264     [fixme] le mieux c'est de faire appel à une bibliothèque spécialisée
265
266     """
267
268     timeId=str(random.randint(1,10000000))+str(time.time())
269
270     return timeId
271
272 def createIdThread():
273     # en prévision d'un version multithreadée
274     return thread.get_ident()
275
276
277 def temporisateur(tpsAttente,delai,incrDelai,pasIncreDelaiBorne = 0.10, \
278                                                 maxPaxIncreDelaiBorne = 1):
279     print "========================"
280     print "nouveau cycle d'attente"
281     print "========================"
282     print "tps d'attente total", tpsAttente
283     print "ancien delai", delai
```

```
284     print "increment du delai", incrDelai
285     print "ancien pas de increment delai ", pasIncreDelaiBorne
286
287
288     if incrDelai < maxPaxIncreDelaiBorne:
289         incrDelai= incrDelai + pasIncreDelaiBorne
290     else :
291         incrDelai= incrDelai + 1
292
293     print "nouveau increment delai ", incrDelai
294
295
296     delai = delai + incrDelai
297
298
299     tpsAttente = tpsAttente + delai
300
301     print "nouveau delai", delai
302     print "nouveau temps d'attente total", tpsAttente
303
304
305     time.sleep(delai)
306
307
308
309     return ([ tpsAttente, delai, incrDelai])
310
311
312
313 def logRequeteDelete():
314     pass
315
316 def logRequeteWrite():
317     pass
318
319
320
321
322
323
```

Typographical Conventions for Python

Python:Normal Text	**Python:Definition Keyword**	Python:Operator
Python:String Substitution	**Python:Command Keyword**	**Python:Flow Control Keyword**
Python:Builtin Function	Python:Special Variable	Python:Extensions
Python:Exceptions	**Python:Overloaders**	Python:Preprocessor
Python:String Char	Python:Long	Python:Float
Python:Int	Python:Hex	Python:Octal
Python:Complex	*Python:Comment*	Python:String
Python:Raw String	**Alerts:Normal Text**	**Alerts:Alert**

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-


__author__ = "Charles Duysinx aka MonsieurCloug"
__copyright__ = "Copyright 2009, Charles Duysinx"
__license__ = "MIT"
__version__ = "0.0"
__revision__ = "$LastChangedRevision: 175 $"[22:-2]
__date__ = "$LastChangedDate: 2009-03-17 16:15:55 +0100 \
           (Mar, 17 Mar 2009) $"[18:-2]


from cPickle import loads, dumps
from math import *

class EvalRedLinda(object):

    """ Cette classe implémente un tuple vivant.

    Pour ce faire nous allons la richesse de Python. En effet, celui-ci
    permet d'interpreter du code python  dans un programme python.

    Python met principalement à la disposition deux instruction exec() et
    execfile() et une fonction eval(). L'exécution peut se faire soit en
    evaluant une chaine (comme un programme d'ailleurs) ou en éxécutant
    du code déjà interpréter. Dans ce dernier cas on utilise la fonction
    compile() qui retourne un objet code. Ce code objet est différent pour
    eval() ou pour exec[file](). Un paramètre permet d'indiquer à compile
    le client final.

    Dans notre implémentation, le type de l'element vivant du tuple est donc
    avant son évaluation de type EvalRedLinda. (un type class donc). Après son
    évaluation il peut être d'un autre type.


    Cette evaluation peut prendre différente forme, puisque le langage Python
    l'autorise. Cela peut-être soit une evaluation d'une expression ou d'un code,
    soit l'exécution d'un bout de code ou d'un fichier de code.

    Donc expression à évaluer se présente sous deux formes :

    la forme fonctionnelle cos (90) +cos (60)
    la forme instruction  a= 10+20. Cette dernière peut être un ou un nombre
    n de lignes.


    ----------------------
```

```
49      La forme fonctionnelle
50      ----------------------
51
52      la forme fonctinnelle est implémentée avec l'instruction eval() de Python.
53      C'est une fonction qui retourne un résultat que nous pouvons directement
54      affecter à une variable.
55      Le contexte d'exécution est le même contexte que le reste du programme.
56      Cela simplifie evidement les choses puisque nous pouvons utiliser directement
57      les bibliothèques, les objets ainsi que toutes les variables globales.
58
59      Si nous voulons isoler ce nouveau appel deux choix s'offre à nous. Soit nous
60      lui passons une copie du scope actuel. Ce qui est de loin la solution la plus
61      facile.
62
63      Soit nous sommes obliger de lui passer un objet compilé dans le cas où nous
64      voudrions utiliser une bibliothque non présente par défaut dans l'interpreteur
65      Python. Dans ce dernier cas nous devons faire appel à la méthode compile pour
66      préparer ce code pour l'expression eval.
67
68      Pour notre part, nous avons choisis de toujours utiliser le contexte actuel.
69
70
71      ----------------------
72      la forme instruction
73      ----------------------
74
75      Cette forme est à proscrire. Car elle n'est signe de bonne pratique de
76      programmeation.En effet, elle se résume à l'exécution d'un
77      programme Python à l'intérieur d'un programme Python.
78      En tout premier lieu, il est nécessaire de lui créer un nouveau contexte.
79      En effet, si ce bout de code reimporte une bibliothèque déjà présente, des
80      erreurs risques de se produire. Réinitialisation intenpestive par exemple.
81
82      Le nouveau contexte est accessible par un dictionnaire.
83
84      Signalons que le debouage de ce type de cette manière de faire est assez
85      délicate puisque soit le code s'exécute dans un contexte isolé et nous nous
86      n'utilisons pas le contexte soit nous récupérons ce contexte et à ce moment là
87      la démarche fonctionnelle est plus simple.
88
89      Pour notre part nous n'utiliseront pas ce type d'approche.
90
91      ------------------------------
92      Illustration des deux approches
93      ------------------------------
94
95
96      Un session interactive montrant l'utilisation de eval et d'exec
97      ---------------------------------------------------------------
98
99      Instanciation d'une variable de classe EvalRedLinda
100     >>> evalOne=None
101     >>> evalOne=EvalRedLinda("somme = 10+20")
102     >>> print "voici le type de evalOne", type(evalOne)
103     voici le type de evalOne <class '__main__.EvalRedLinda'>
104
105     Ici nous allons commencer par tester la forme instruction
106     c'est à dire exec. Comme cela est illustrer dans le code, j'utilise
107     un nouveau contexte en déclarant un dictionnaire pour mon scope
```

```
global. Après exécution, j'ai accès à toutes les variables du contexte
d'exécution via le dictionnaire monDico.
Ici la variable somme est accessible via le dictionnaire que nous avons
passé en argument à notre instruction exec.


>>>  # utilisation de l'instruction exec

>>> monDico={}
>>> exec(evalOne.expression,monDico)
>>> # pour accéder à la variable résultat

>>> print "voici l'evaluation de la variable somme ",monDico['somme']
voici l'evaluation de la variable somme  30


Voici maintenant, la même instruction qui s'applique ici à un fichier
tableMultiplication.py qui est un fichier python.


from math import *

unNombre= 12
uneListe =[]
for compteur in xrange(0,unNombre+1):
    uneListe.append(compteur * unNombre)

monSinus = sin(90)

Voici la session interactive :


>>> # utilisation de l'instruction execfile

>>> execfile("tableMutiplication.py",monDico)
>>> print "voici la liste", monDico['uneListe']
voici la liste [0, 12, 24, 36, 48, 60, 72, 84, 96, 108, 120, 132, 144]
>>> print "voici le sinus", monDico['monSinus']
voici le sinus 0.893996663601

Comme nous pouvons le constater cette dernière manière de faire
n'est pas vraiment souple par rapport par exemple à l'orienté objet.
Il n'est vraiment pas recommandable d'utiliser cette manière de faire.


En fin voici, la manière fonctionnelle. D'abord éxecutée dans un
nouveau contexte, simplemen en duplicant le contexte courrant à l'aide
de la methode globals.


>>> # utilisation dans un nouveau contexte d'exécution
>>> evalOne = None
>>> evalOne = EvalRedLinda("cos(90)+cos(0)")
>>> monDico=globals()
>>> resultatExpression = eval(evalOne.expression,monDico)
>>> print "voici le résultat de mon expression",resultatExpression
voici le résultat de mon expression 0.551926383871
```

```
167     Et ensuite en l'exécutant dans le contexte courrant.
168     Cette manière de faire est en définitive la plus souple et permet
169     un meilleur moyen de contrôle sur ce que l'on fait. Pour ces raisons
170     nous utiliserons principalement cette méthode.
171
172
173     >>> eval0ne.expression="cos(90)+cos(0)"
174     >>> # utilisation de la fonction eval dans le même scope
175
176     >>> resultatExpression=eval(eval0ne.expression)
177     >>> print "voici le résultat de mon expression",resultatExpression
178     voici le résultat de mon expression 0.551926383871
179
180
181     ---------------------------------
182     L'implémentation proprement dite
183     ---------------------------------
184
185     Le mécanisme que nous utilisons est le suivant :
186         lors de l'enregistrement de l'élement vivant dans le tupleSpace,
187         l'expression n'est pas évaluée. Cette évaluation sera éxécutée
188         lors de sa lecture ou retrait du tupleSpace.
189
190     Tous nos objets sont sérialisés lors de leur stockage sur le tupleSpace.
191     Lors de leur désérialisation nous appelons une seconde fois la méthode init
192     qui va provaquer l'interpretation de la variable membre expression.
193
194     Pour mettre en place ce mécanisme nous avons besoin d'un drapeau evalLiveTuple
195     qui est faux lors de l'initialisation de la classe et prend l'état de Vrai après
196     initialisation.
197
198     Lors de la seconde initialisation c'est ce drapeau qui provoque l'évaluation.
199
200     Après évaluation la variable membre expression contient toujours, l'expression
201     à évaluer. Cette information permet un contrôle.
202
203
204     Cette classe utilise les paramètres suivants :
205
206     expression          : qui contient soit l'instruction soit la fonction a évaluer
207
208
209     isFile=False        : qui indique si l'expression est un fichier
210
211
212     typeEval=True       : qui est à Vrai pour la forme fonctionnelle et faux pour
213                           la forme instruction
214
215
216     globalDico=None     : Permet de définir le contexte global (scope)
217
218
219     localDico=None      : Permet de définir le contexte global (scope)
220
221
222     evalLiveTuple=False : force l'évaluation de l'expression.
223
224
225     Après évaluation de l'expression, les valeurs sont accessibles de manière
```

```
226     différente selon la forme de l'expression :
227
228
229         a) forme fonctionnelle :
230
231             la valeur se trouve dans la variable membre expressionEvaluated
232             Le contexte est par défaut le contexte courrant. Mais soulignons
233             toutefois que ce contexte n'est pas accessible dans le scope
234             global. En effet, l'évaluation se fait bien au niveau de la classe
235             et non au niveau global.
236
237         b) la forme instruction :
238
239             la ou les valeurs se trouves dans les deux dictionnaires des
240             variables membres globalDico et localDico.
241
242             C'est le dictionnaire localDico qui contient en fait le contexte
243             global de l'exécution alors que globalDico contient le contexte
244             global lors de l'appel de l'exécution.
245
246             Nous retrouverons donc nos valeurs dans le dictionnaire localDico
247
248             Attirons l'attention que seul les valeurs connue au premier niveau
249             d'exécution sont accessibles. On peut comprendre dès lors que le
250             déboguage est une véritable partie de plaisir puisque nous avons
251             en fait une boite noire.
252
253
254             Remarque importante :
255             ---------------------
256
257             L'évaluation se fait dans le contexte que l'on passe à l'initialisation
258             de la classe. Si aucun contexte n'est passé, c'est avec un dictionnaire
259             vide pour le contexte que la classe est initialisée.
260             Sans cela, il est impossible d'avoir aux valeurs manipulées par le code
261             exécuté lors de l'évaluation.
262
263             Ce aspect des choses, fait qu'il vaut mieux considérer ce type
264             d'utilisation comme une "procédure à l'ancienne". Même si la langage
265             Python ne fait pas la différence.
266
267             On peut si l'on veut faire un copie du contexte courrant. Cette manière
268             de faire est une très mauvaise idée comme nous l'avons  souligné
269             précédement. La duplication d'import de classe est source d'erreur.
270
271             Il vaut mieux donc d'isoler le plus possible le code que l'on va
    exécuter
272             via cette stratégie.
273
274
275
276
277     Cas d'utilisation de la classe EvalRedLinda
278     -------------------------------------------
279
280     Les paramètres
281     **************
282
283     expression : qui contient soit l'instruction soit la fonction a évaluer
```

```
284
285
286     isFile=False : qui indique si l'expression est un fichier
287
288
289     typeEval=True : qui est à Vrai pour la forme fonctionnelle et faux pour
290                     la forme instruction
291
292
293     globalDico={} : qui est le contexte global. Si le dictionnaire est vide
294                     c'est le contexte courrant qui est utilisé.
295
296
297     localDico={}  : qui est le contexte local. Si le dictionnaire est vide
298                     c'est le contexte courrant qui est utilisé.
299
300     evalLiveTuple=False : force l'évaluation de l'expression.
301
302     Les variales membres
303     ********************
304
305     expressionEvaluated = "" dans la forme fonctionnelle l'expression évaluée
306
307
308
309     Imitation d'exec dans le  contexte par défaut soit un contexte vide
310     ***********************************************************************
311     >>> monEval= None
312     >>> expression = "somme = 22+44"
313     >>> isFichier = False
314     >>> typeEval = False
315     >>> globalDico = None
316     >>> localDico = None
317     >>> evalLiveTuple = True
318     >>> monEval = EvalRedLinda(expression, \
319                                isFichier, \
320                                typeEval,  \
321                                globalDico, \
322                                localDico, \
323                                evalLiveTuple)
324     >>> print "variable membre expression ", monEval.expression
325     variable membre expression  somme = 22+44
326
327     >>> print "voici le contenu de localDico", monEval.localDico
328     voici le contenu de localDico {'somme': 66}
329
330
331     Imitation d'exec avec un nouveau contexte par copie du contexte courant
332     ***********************************************************************
333     >>> monEval= None
334     >>> expression = "somme = 22+44"
335     >>> isFichier = False
336     >>> typeEval = False
337     >>> globalDico = globals()
338     >>> localDico = locals()
339     >>> evalLiveTuple = True
340     >>> monEval = EvalRedLinda(expression, \
341                                isFichier, \
342                                typeEval,  \
```

```
343                                 globalDico, \
344                                 localDico, \
345                                 evalLiveTuple)
346     >>> print "variable membre expression ", monEval.expression
347     variable membre expression  somme = 22+44
348
349     >>> print "voici la valeur de la clé somme dans localDico",
    monEval.localDico['somme']
350     voici la valeur de la clé somme dans localDico 66
351
352
353     Imitation d'execfile avec un nouveau contexte par copie du contexte courant
354     ***************************************************************************
355     >>> monEval= None
356     >>> expression = "tableMutiplication.py"
357     >>> isFichier = True
358     >>> typeEval = False
359     >>> globalDico = globals()
360     >>> localDico = locals()
361     >>> evalLiveTuple = True
362     >>> monEval = EvalRedLinda(expression, \
363                                 isFichier, \
364                                 typeEval,  \
365                                 globalDico, \
366                                 localDico, \
367                                 evalLiveTuple)
368     >>> print "variable membre expression ", monEval.expression
369     variable membre expression  tableMutiplication.py
370
371
372     >>> print "voici la valeur de la clé uneListe dans localDico",
    monEval.localDico["uneListe"]
373     voici la valeur de la clé uneListe dans localDico [0, 12, 24, 36, 48, 60, 72,
    84, 96, 108, 120, 132, 144]
374
375
376
377
378
379     Imitation d'execfile dans le  contexte par défaut soit un contexte vide
380     ***************************************************************************
381     >>> monEval= None
382     >>> expression = "tableMutiplication.py"
383     >>> isFichier = True
384     >>> typeEval = False
385     >>> globalDico = None
386     >>> localDico = None
387     >>> evalLiveTuple = True
388     >>> monEval = EvalRedLinda(expression, \
389                                 isFichier, \
390                                 typeEval,  \
391                                 globalDico, \
392                                 localDico, \
393                                 evalLiveTuple)
394     >>> print "variable membre expression ", monEval.expression
395     variable membre expression  tableMutiplication.py
396
397     >>> print "voici la valeur de la clé uneListe dans localDico",
    monEval.localDico["uneListe"]
```

```
398     voici la valeur de la clé uneListe dans localDico [0, 12, 24, 36, 48, 60, 72,
    84, 96, 108, 120, 132, 144]
399
400
401
402     Imitation d'eval dans le même contexte
403     ***************************************
404     >>> monEval= None
405     >>> expression = "cos(90)+cos(0)"
406     >>> isFichier = False
407     >>> typeEval = True
408     >>> globalDico = None
409     >>> localDico = None
410     >>> evalLiveTuple = True
411     >>> monEval = EvalRedLinda(expression, \
412                                 isFichier, \
413                                 typeEval,  \
414                                 globalDico, \
415                                 localDico, \
416                                 evalLiveTuple)
417     >>> print "variable membre expression ", monEval.expression
418     variable membre expression  cos(90)+cos(0)
419
420     >>> print "voici le résultat de mon expression",monEval.expressionEvaluated
421     voici le résultat de mon expression 0.551926383871
422
423
424
425     Imitation d'eval avec copie du contexte
426     ****************************************
427     >>> monEval= None
428     >>> expression = "cos(90)+cos(0)"
429     >>> isFichier = False
430     >>> typeEval = True
431     >>> globalDico = globals()
432     >>> localDico = locals()
433     >>> evalLiveTuple = True
434     >>> monEval = EvalRedLinda(expression, \
435                                 isFichier, \
436                                 typeEval,  \
437                                 globalDico, \
438                                 localDico, \
439                                 evalLiveTuple)
440     >>> print "variable membre expression ", monEval.expression
441     variable membre expression  cos(90)+cos(0)
442
443     >>> print "voici le résultat de mon expression",monEval.expressionEvaluated
444     voici le résultat de mon expression 0.551926383871
445
446
447
448     Evaluation directe après initialisation
449     ****************************************
450     >>> monEval= None
451     >>> expression = "cos(90)+cos(0)"
452     >>> isFichier = False
453     >>> typeEval = True
454     >>> globalDico = globals()
455     >>> localDico = locals()
```

```
456     >>> evalLiveTuple = False
457     >>> monEval = EvalRedLinda(expression, \
458                                 isFichier, \
459                                 typeEval,  \
460                                 globalDico, \
461                                 localDico, \
462                                 evalLiveTuple)
463
464     >>> print "variable membre expression ", monEval.expression
465     variable membre expression  cos(90)+cos(0)
466
467     >>> print "la variable membre expressionEvaluated est bien",
    monEval.expressionEvaluated
468     la variable membre expressionEvaluated est bien None
469
470     >>> monEval.init()
471
472
473     >>> print "voici le résultat de mon expression après l'exécution de init()",
    monEval.expressionEvaluated
474     voici le résultat de mon expression après l'exécution de init() 0.551926383871
475
476
477
478     -------------
479     En conclusion
480     ------------- .
481
482     Notre classe EvalRedLinda, implémente les deux manières de faire
483     quant à l'évalution d'une expression de Python. Que l'on veuille
484     utilser exec ou eval, ce n'est pas à nous à restreindre
485     la richesse de Python. En effet,notre classe ne fait
486     qu'ajouter une couche d'abstraction pour
487     permette à RedLinda de procposer les tuples vivants.
488
489
490
491     -------
492     Doctest
493     -------
494
495     Pour vérifier le fonctionnement de cette classe. Nous avons choisi
496     d'utiliser le développement guidé par la documentation.
497     Cela nous parraissait adapté dans ce cas-ci. En effet, cette classe
498     a peu de dépendance. Nous voulions aussi expliquer les mécanismes que
499     nous utilisons. Cela nous obligeait à écrire de la documentation. Et
500     illustrer notre propos par des exemples concrets de cas d'utilisation.
501     Les Doctest que propose Python était donc totalement adapté dans ce
502     cas-ci. <FIXME> Expliquer et donner des références pour les doctests.
503
504
505     Quant à son utilisatio pratique remaquons que les facilités offertes
506     aux programmeurs sont loins d'une utilisation facile.Par exemple dans
507     ce bout de log on aimerait au moins connaître la ligne ou l'erreur
508     s'est produite. Cela oblige toutefois à écrire les tests un par un et à
509     les tester.
510     Maleureusement, en cas de changement on imagine facilement le travail
511     fastitieux que cela génére surtout si c'était test sont simplement
512     des copié-coller qui teste plusieurs cas avec les mêmes messages. Ce
```

```
513     qui en fait de compte est assez courrant dans un développement.
514
515     Par contre, on peut garantir que le docuemntation est bien l'illustration
516     de l'implémentation. Tous les exemples donnés correpondent bien à ce que
517     l'on aurra si on test de manière interactive.
518
519     Ajoutons, que les exemples choisis devraient être avec des valeurs cible.
520     Ici par exemple nous traitons une fonction qui n'est pas dans le contexte
521     courrant de l'interpreteur Python.
522
523
524
525     File "__main__", line ?, in __main__.EvalRedLinda
526     Failed example:
527     print "voici le résultat de mon expression", monEval.expressionEvaluated
528     Expected nothing
529     Got:
530     voici le résultat de mon expression 0.551926383871
531
532
533     """
534
535
536     __safe_for_unpickling__ = True
537
538     expression = None             # mis à None pour pouvoir utiliser les docTest
539     evalLiveTuple = False         #
540     expressionEvaluated = None    # mis à None pour pouvoir utiliser les docTest
541     liveTupleEvaluated = False
542
543     def __init__(self,expression,isFichier=False,typeEval=True,\
544                  globalDico=None,localDico=None,evalLiveTuple=False):
545
546         self.expression = expression
547         self.isFichier= isFichier
548         self.typeEval = typeEval
549
550         ## configuration de l'enviromment gloabal
551         if self.typeEval:
552             """ forme eval = initialisation du contexte"""
553             self.globalDico=globalDico
554
555
556         elif not self.typeEval:
557             # format instruction
558             if globalDico:
559                 # nouveau contexte
560
561                 self.globalDico=globalDico
562             else :
563                 # nouveau contexte par defaut
564                 self.globalDico={}
565
566         ## configuration de l'enviromment local
567
568         if self.typeEval:
569             """ forme eval = initialisation du contexte"""
570             self.localDico=localDico
571
```

```
572         elif not self.typeEval:
573             # format instruction
574             if localDico:
575                 # nouveau contexte
576 
577                 self.localDico=localDico
578             else :
579                 # nouveau contexte par defaut
580                 self.localDico={}
581 
582 
583         self.evalLiveTuple = evalLiveTuple
584         if self.evalLiveTuple:
585             self.liveTupleEvaluated = True
586 
587 
588 
589             if self.isFichier:
590                 execfile(self.expression,self.globalDico, self.localDico)
591 
592             elif self.typeEval:
593                 self.expressionEvaluated = eval(self.expression,self.globalDico,
    self.localDico)
594 
595             elif (not self.typeEval) and (not self.isFichier):
596 
597                 exec(self.expression,self.globalDico, self.localDico)
598 
599 
600             else:
601                 "on est dans qui ne devrait jamais arriver "
602                 self.expressionEvaluated = None
603 
604         ### changement du flag pour provoquer l'evaluation
605 
606         self.evalLiveTuple = True
607 
608 
609     def __getstate__ (self):
610         """obligatoire si nous voulons que la classe soit pickelisable
611         mais j'ai quand même un gros soucis. Car j'ai un o
612 
613         """
614 
615         etatClasse = { "evalLiveTuple" : self.evalLiveTuple, \
616                        "expression" : self.expression, \
617                        "isFichier" :self.isFichier, \
618                        "typeEval" : self.typeEval, \
619                        "globalDico" : self.globalDico, \
620                        "localDico" : self.localDico , \
621                        "evalLiveTuple" :self.evalLiveTuple}
622 
623 
624         return etatClasse
625 
626 
627     def __setstate__(self,dico):
628 
629 
```

```
630         self.__init__(dico['expression'], \
631                       dico['isFichier'], \
632                       dico['typeEval'], \
633                       dico['globalDico'], \
634                       dico['localDico'], \
635                       dico['evalLiveTuple'])
636
637
638     def __getnewargs__(self,dico):
639
640         return (dico['expression'], \
641                 dico['isFichier'], \
642                 dico['typeEval'], \
643                 dico['globalDico'], \
644                 dico['localDico'])
645
646
647     def __str__(self):
648         """ Sa réprésenation
649         """
650
651         if self.liveTupleEvaluated:
652             return "<Live Tuple Evaluated : %s >" % (self.expression)
653         else:
654             return "<Live Tuple For Evaluation : %s >" % (self.expression)
655
656
657     def __repr__(self):
658         """ Sa réprésenation
659         """
660
661         if self.liveTupleEvaluated:
662             return "<Live Tuple  Evaluated : %s >" % (self.expression)
663         else:
664             return "<Live Tuple For Evaluation : %s >" % (self.expression)
665
666
667     def init(self):
668         """cette méthode est implémentée pour le test unitaire. Elle permet
669         en effet, de tester que l'evaluation provoquée par la lecture du tuple
670         est la même que celle qui est faite en "local"
671
672         """
673         dico = self.__dict__
674         self.__init__(dico['expression'], \
675                       dico['isFichier'], \
676                       dico['typeEval'], \
677                       dico['globalDico'], \
678                       dico['localDico'], \
679                       dico['evalLiveTuple'])
680
681
682
683 if __name__ == '__main__':
684
685     import doctest
686     doctest.testmod()
687
688
```

689
690
691
692
693

Typographical Conventions for Python

Python:Normal Text	**Python:Definition Keyword**	Python:Operator
Python:String Substitution	**Python:Command Keyword**	**Python:Flow Control Keyword**
Python:Builtin Function	Python:Special Variable	Python:Extensions
Python:Exceptions	**Python:Overloaders**	Python:Preprocessor
Python:String Char	Python:Long	Python:Float
Python:Int	Python:Hex	Python:Octal
Python:Complex	*Python:Comment*	Python:String
Python:Raw String	Alerts:Normal Text	**Alerts:Alert**

```
1 from math import *
2
3 unNombre= 12
4 uneListe =[]
5 for compteur in xrange(0,unNombre+1):
6     uneListe.append(compteur * unNombre)
7
8 monSinus = sin(90)
9
```

Typographical Conventions for Python

Python:Normal Text	**Python:Definition Keyword**	Python:Operator
Python:String Substitution	**Python:Command Keyword**	**Python:Flow Control Keyword**
Python:Builtin Function	Python:Special Variable	Python:Extensions
Python:Exceptions	**Python:Overloaders**	Python:Preprocessor
Python:String Char	Python:Long	Python:Float
Python:Int	Python:Hex	Python:Octal
Python:Complex	*Python:Comment*	Python:String
Python:Raw String	**Alerts:Normal Text**	**Alerts:Alert**

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-


__author__ = "Charles Duysinx aka MonsieurCloug"
__copyright__ = "Copyright 2009, Charles Duysinx"
__license__ = "MIT"
__version__ = "0.0"
__revision__ = "$LastChangedRevision: 175 $"[22:-2]
__date__ = "$LastChangedDate: 2009-03-17 16:15:55 +0100 \
           (Mar, 17 Mar 2009) $"[18:-2]



import sys
sys.path.append('/home/gdh2/Documents/memoire/workspace/redlindalock/src/')
import kernel as linda
#import linda
from nose.tools import *
import random

##############################################################################
#                                                                            #
#               véfifier que le tuple(avec un seul élement) fonctionne       #
#                                                                            #
##############################################################################


##############################################################################
#                                                                            #
#               Test pour tupleSpace <fixMe> il faudra la scinder            #
#                                                                            #
##############################################################################


def testCreationTupleSpace():
    """
    Initialisation de la classe test

    """

    # creation de linda universe

    linda.connect()

    # on verifie que c'est un type tuple space"
    assert str(linda.universe)=='<Universal Tuplespace>'

```

```
49
50      # on verifie que son id est bien 0!0
51      assert linda.universe._id=='0!0'
52
53      # on vérifie que la classe a été détruite
54
55      "creation du cleaner"
56      cleaner=linda.TupleSpace()
57
58      # on vide Redis
59      cleaner.flush()
60
61      # on verifie que la base est bien vide
62      assert cleaner.dbsize()==0
63
64      # creation de origine
65      origine=linda.TupleSpace()
66      linda.universe._out(("origine",origine))
67      # attention après creation il y un compteur de tupleSpace
68      assert cleaner.dbsize()==2
69
70
71      # creation de destination
72      destination=linda.TupleSpace()
73      linda.universe._out(("destination",destination))
74      assert cleaner.dbsize()==3
75
76      for element in xrange (0,10):
77          # str int float
78          origine._out(("coucou",element,(element *10.0)  ))
79
80      assert cleaner.dbsize()==13
81
82      """suppression de la mémoire des deux classes cela
83       est nécessaire si on veut les instanciés à nouveau.
84
85      """
86
87
88      origine = None
89      destination = None
90
91      # on verifie la suppression superflu mais on ne sait j0amais....
92      assert origine == None
93      assert destination == None
94
95
96      # récupération de origine
97      origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
98      #on vérifie que l'initialisation est bien faite
99
100     assert origine._id=='0!1'
101     assert origine.delaiVerrouTransactionServeur >= 1
102     assert origine.delaiVerrouGlobalServeur >=1
103     assert origine.delaiUnblockRequete >=1
104     assert origine.delaiBlockRequete >= 1
105     assert origine.logRequete <> None
106     assert origine.host <> None
107     assert origine.port == 6379
```

```
108     assert origine.nodelay == None
109     assert origine.charset == 'utf8'
110     assert origine.errors == 'strict'
111     assert origine._sock <> None
112     assert origine._fp <> None
113     assert (origine.db >= 0) and (origine.db <= 10)
114
115
116     # il faut vérifier que cette classe est complète
117
118     # récupération de destination
119     destination = linda.universe._rd(("destination",linda.TupleSpace))[1]
120
121     assert destination._id == '0!2'
122     # pas besoin de vérifier que cette classe est bien un tupleSpace.
123
124
125     assert origine.dbsize()==13
126
127
128
129 ##########################################################################################
130 #                                                                                        #
131 #                              Test pour le in                                           #
132 #                                                                                        #
133 ##########################################################################################
134
135
136
137 def testInConsommation():
138     # je prends 5 élements d'origine sur 10
139     # dont out(("coucou",element,(element *10.0)  ))
140     origine=None
141     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
142
143     for compteur in xrange(0,5):
144         origine._in((str,int, float))
145
146     assert origine.dbsize()==8
147
148     #test sur le type
149     for compteur in xrange(0,5):
150         origine._in((str,int, float))
151
152     assert origine.dbsize()==3
153
154
155 def testInTypeAntiTuple():
156     "verifie que l'on récupère un tuple donné par l'antituple"
157     origine = None
158     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
159     tupleSend=("testInTypeAntiTuple",1,1.2,[1],(1,2))
160     origine._out(tupleSend)
161     tupleReceive=origine._in((str,int,float,list,tuple))
162     assert tupleSend == tupleReceive
163
164     # verfication que c'était bien celui-là que l'on devait récupérer
165
166     tupleVide = origine._rdp(tupleSend)
```

```
167
168     assert tupleVide ==()
169
170
171 def testInTypeTuple():
172     "verifie que l'on récupère le tuple que l'on mis sur le tupleSpace"
173     origine = None
174     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
175     tupleSend=("testInTypeTuple",1,1.2,[1],(1,2))
176     origine._out(tupleSend)
177     tupleReceive=origine._in(("testInTypeTuple",1,1.2,[1],(1,2)))
178     assert tupleSend == tupleReceive
179
180     # verfication que c'était bien celui-là que l'on devait récupérer
181
182     tupleVide = origine._rdp(tupleSend)
183
184     assert tupleVide ==()
185
186
187
188
189 ##############################################################################
190 #                                                                            #
191 #                          Test pour out                                     #
192 #                                                                            #
193 ##############################################################################
194
195
196 def testOutSizeDbOk():
197     origine=None
198     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
199
200     # nombre d'enregistrements de la base Redis
201     dbSizeDebut = origine.dbsize()
202
203     # je rajoute cinq tuples "identiques"
204     for compteur in xrange(0,5):
205         origine._out(("test",1, 3.14))
206     dbSizeFin = origine.dbsize()
207
208     # je vérifie qu'ils sont ajoutés
209     assert      dbSizeFin == dbSizeDebut +5
210
211     # je rajoute cinq tuples "différents"
212     dbSizeDebut = origine.dbsize()
213
214     for compteur in xrange(0,5):
215         origine._out((str(compteur),compteur, compteur *1.0))
216
217     dbSizeFin = origine.dbsize()
218
219     assert dbSizeFin == dbSizeDebut +5
220
221
222 ##############################################################################
223 #                                                                            #
224 #                          Test pour le rd                                   #
225 #                                                                            #
```

```
226 ##############################################################################
227
228
229
230 def testRdNonConsommation():
231     "verifie que l'on ne consomme pas l'élément"
232     origine=None
233     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
234
235     # nombre d'enregistrements de la base Redis
236     dbSizeDebut = origine.dbsize()
237
238     # je lis cinq fois le même tuple"
239     for compteur in xrange(0,5):
240         origine._rd(("test",1, 3.14))
241     dbSizeFin = origine.dbsize()
242
243     # je vérifie que la taille de la base n'a pas changé
244     assert      dbSizeFin == dbSizeDebut
245
246     # je lis cinq tuples "différents"
247
248     for compteur in xrange(0,5):
249         origine._rd((str,int,float))
250
251     dbSizeFin = origine.dbsize()
252
253     assert dbSizeFin == dbSizeDebut
254
255
256 def testRdTypeAntiTuple():
257     "verifie que l'on récupère le bon tuple par une requete antituple"
258     origine = None
259     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
260     tupleSend=("testRdTypeAntiTuple",1,1.2,[1],(1,2))
261     origine._out(tupleSend)
262     tupleReceive=origine._rd((str,int,float,list,tuple))
263     assert tupleSend == tupleReceive
264
265     # attention destruction et verfication
266     origine._in(tupleSend)
267     tupleVide = origine._rdp(tupleSend)
268
269     assert tupleVide ==()
270
271 def testRdTypeTuple():
272     "verifie que l'on récupère un tuple par sa requete tuple"
273     origine = None
274     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
275     tupleSend=("testRdTypeTuple",1,1.2,[1],(1,2))
276     origine._out(tupleSend)
277     tupleReceive=origine._rd(("testRdTypeTuple",1,1.2,[1],(1,2)))
278     assert tupleSend == tupleReceive
279
280     # attention destruction et verfication
281     origine._in(tupleSend)
282     tupleVide = origine._rdp(tupleSend)
283
284     assert tupleVide ==()
```

```
285
286
287
288
289 ##############################################################################
290 #                                                                            #
291 #                          Test pour le rdp                                  #
292 #                                                                            #
293 ##############################################################################
294
295
296
297 def testRdNonConsommation():
298     "verifie que l'on ne consomme pas l'élément"
299     origine=None
300     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
301
302     # nombre d'enregistrements de la base Redis
303     dbSizeDebut = origine.dbsize()
304
305     # je lis cinq fois le même tuple"
306     for compteur in xrange(0,5):
307         origine._rdp(("test",1, 3.14))
308     dbSizeFin = origine.dbsize()
309
310     # je vérifie que la taille de la base n'a pas changé
311     assert      dbSizeFin == dbSizeDebut
312
313     # je lis cinq tuples "différents"
314
315     for compteur in xrange(0,5):
316         origine._rdp((str,int,float))
317
318     dbSizeFin = origine.dbsize()
319
320     assert      dbSizeFin == dbSizeDebut
321
322
323 def testRdpTypeAntiTuple():
324     "verifie que l'on récupère le bon tuple par une requete antituple"
325     origine = None
326     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
327     tupleSend=("testRdpTypeAntiTuple",1,1.2,[1],(1,2))
328     origine._out(tupleSend)
329     tupleReceive=origine._rdp((str,int,float,list,tuple))
330     assert tupleSend == tupleReceive
331
332     # attention destruction et verfication
333     origine._in(tupleSend)
334     tupleVide = origine._rdp(tupleSend)
335
336     assert tupleVide ==()
337
338
339 def testRdpTypeTuple():
340     "verifie que l'on récupère un tuple formé des mêmes élements et de même
    longueur"
341     origine = None
342     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
```

```
343     tupleSend=("testRdpTypeTuple",1,1.2,[1],(1,2))
344     origine._out(tupleSend)
345     tupleReceive=origine._rdp(("testRdpTypeTuple",1,1.2,[1],(1,2)))
346     assert tupleSend == tupleReceive
347
348     # attention destruction et verfication
349     origine._in(tupleSend)
350     tupleVide = origine._rdp(tupleSend)
351
352     assert tupleVide ==()
353
354
355
356
357 def testRdpNonPresent():
358     "verifie que l'on a bien un tuple vide si le tuple n'existe pas"
359     origine=None
360     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
361
362     # demande d'un tuple inexistant
363
364     tupleVide=origine._rdp(("testRdNonPresent","testRdNonPresent", 1))
365
366     assert tupleVide==()
367
368
369 ##########################################################################################
370 #                                                                                        #
371 #                              Test pour le inp                                          #
372 #                                                                                        #
373 ##########################################################################################
374
375
376
377 def testInpConsommation():
378     " test si la directive consomme les tuples"
379
380     origine=None
381     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
382     dbSizeDebut = origine.dbsize()
383
384     # je mets cinq tuples dans le tupleSpace
385
386     for element in xrange(0,5):
387         origine._out((element *11.55,element *4,"aurevoir" ))
388
389     # je retire cinq tuples
390     for compteur in xrange(0,5):
391         origine._inp((str,int, float))
392
393     dbSizeFin = origine.dbsize()
394
395     assert dbSizeDebut == dbSizeFin
396
397 def testInpTypeAntiTuple():
398     "verifie que l'on récupère le bon tuple par une requete antituple"
399     origine = None
400     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
401     tupleSend=("testInpTypeAntiTuple",1,1.2,[1],(1,2))
```

```
402     origine._out(tupleSend)
403     tupleReceive=origine._in((str,int,float,list,tuple))
404     assert tupleSend == tupleReceive
405
406     # verfication que c'était bien celui-là que l'on devait récupérer
407
408     tupleVide = origine._rdp(tupleSend)
409
410     assert tupleVide ==()
411
412 def testInpTypeTuple():
413     "verifie que l'on récupère un tuple formé des mêmes élements et de même
    longueur"
414     origine = None
415     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
416     tupleSend=("testInpTypeTuple",1,1.2,[1],(1,2))
417     origine._out(tupleSend)
418     tupleReceive=origine._in(("testInpTypeTuple",1,1.2,[1],(1,2)))
419     assert tupleSend == tupleReceive
420
421     # verfication que c'était bien celui-là que l'on devait récupérer
422
423     tupleVide = origine._rdp(tupleSend)
424
425     assert tupleVide ==()
426
427
428
429
430 ##########################################################################
431 #                                                                        #
432 #                        Test pour le copy_collect                       #
433 #                                                                        #
434 ##########################################################################
435
436
437 def testCopyCollectTupleNonPresent():
438     "test si la directive retourne 0 quand il n'y pas de matching"
439
440     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
441     destination = linda.universe._rd(("destination",linda.TupleSpace))[1]
442
443     ## test d'une copie d'élement qui n'existe pas
444
445     nb = origine.copy_collect(destination,("testCopyCollectTupleNonPresent",int))
446
447     assert nb==0
448
449
450
451 def testCopyCollectReturnSize():
452     "test si la directive retourne le nombre exact d'élément copiés"
453
454
455     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
456     destination = linda.universe._rd(("destination",linda.TupleSpace))[1]
457
458     # creation sur le tupleSpace origine
459     for element in xrange(0, 10):
```

```
460         aleaOne = random.randint(1,1000)
461         aleaTwo = random.random()
462         origine._out(("testCopyCollectReturnSize",aleaOne, aleaTwo))
463 
464     nb = origine.copy_collect(destination,("testCopyCollectReturnSize",int, float))
465 
466     assert nb==10
467 
468 
469 def testCopyCollectSizeDb():
470 
471     "test si la base retourne le nombre exact d'élément copiés"
472 
473     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
474     destination = linda.universe._rd(("destination",linda.TupleSpace))[1]
475 
476     dbSizeDebut = origine.dbsize()
477 
478     # creation sur le tupleSpace origine
479     for element in xrange(0, 10):
480         aleaOne = random.randint(1,1000)
481         aleaTwo = random.random()
482         origine._out(("testCopyCollectSizeDb",aleaOne, aleaTwo))
483 
484     nb = origine.copy_collect(destination,("testCopyCollectSizeDb",int, float))
485     dbSizeFin = origine.dbsize()
486 
487     assert dbSizeFin == dbSizeDebut + 20
488 
489 
490 def testCopyCollectRejeux():
491     "test si le rejeux"
492 
493     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
494     destination = linda.universe._rd(("destination",linda.TupleSpace))[1]
495 
496     dbSizeDebut = origine.dbsize()
497 
498     # creation sur le tupleSpace origine
499     for element in xrange(0, 10):
500         aleaOne = random.randint(1,1000)
501         aleaTwo = random.random()
502         origine._out(("testCopyCollectRejeux",aleaOne, aleaTwo))
503 
504     nb = origine.copy_collect(destination,("testCopyCollectRejeux",int, float))
505 
506     # rejeux
507     nb = origine.copy_collect(destination,("testCopyCollectRejeux",int, float))
508 
509 
510     dbSizeFin = origine.dbsize()
511 
512     assert dbSizeFin == dbSizeDebut +20
513 
514 
515 def testCopyCollectNonConsommation():
516     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
517     destination = linda.universe._rd(("destination",linda.TupleSpace))[1]
518 
```

```
519
520     # creation sur le tupleSpace origine
521     for element in xrange(0, 10):
522         aleaOne = random.randint(1,1000)
523         aleaTwo = random.random()
524         origine._out(("testCopyCollectNonConsommation",aleaOne, aleaTwo))
525
526     nb = origine.copy_collect(destination,("testCopyCollectNonConsommation",int,
    float))
527
528     # rejeux
529     nb = origine.copy_collect(destination,("testCopyCollectNonConsommation",int,
    float))
530
531
532     assert nb == 10
533
534
535 def testCopyCollectDestination():
536     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
537     destinationSecrete = linda.TupleSpace()
538
539     # creation sur le tupleSpace origine
540     for element in xrange(0, 10):
541         aleaOne = random.randint(1,1000)
542         aleaTwo = random.random()
543         origine._out(("testCopyCollectDestination",aleaOne, aleaTwo))
544
545     nb = origine.copy_collect(destinationSecrete,("testCopyCollectDestination",int,
    float))
546
547     #on cosomme
548     for compteur in range (0,10):
549         destinationSecrete._in(("testCopyCollectDestination",int, float))
550
551     tupleVide = destinationSecrete._rdp(("testCopyCollectDestination",int, float))
552
553     assert tupleVide == ()
554
555
556 ##############################################################################
557 #                                                                            #
558 #                          Test pour le Collect                              #
559 #                                                                            #
560 ##############################################################################
561
562 def testCollectTupleNonPresent():
563     "test si la directive retourne 0 quand il n'y pas de matching"
564
565     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
566     destination = linda.universe._rd(("destination",linda.TupleSpace))[1]
567
568     ## test d'une copie d'élement qui n'existe pas
569
570     nb = origine.collect(destination,("bonjour",int,"toto le heros"))
571
572     assert nb==0
573
574
```

```
575 def testCollectReturnSize():
576     "test si la directive retourne le nombre exact d'élément déplacés"
577 
578     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
579     destination = linda.universe._rd(("destination",linda.TupleSpace))[1]
580 
581     # creation sur le tupleSpace origine
582     for element in xrange(0, 10):
583         aleaOne = random.randint(1,1000)
584         aleaTwo = random.random()
585         origine._out(("testCopie",aleaOne, aleaTwo))
586 
587     nb = origine.collect(destination,("testCopie",int, float))
588 
589     assert nb==10
590 
591 
592 def testCollectSizeDb():
593 
594     "test si la base retourne le nombre exact d'élément copiés"
595 
596     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
597     destination = linda.universe._rd(("destination",linda.TupleSpace))[1]
598 
599     dbSizeDebut = origine.dbsize()
600 
601     # creation sur le tupleSpace origine
602     for element in xrange(0, 10):
603         aleaOne = random.randint(1,1000)
604         aleaTwo = random.random()
605         origine._out(("testCopy_CollectSizeDb",aleaOne, aleaTwo))
606 
607     nb = origine.collect(destination,("testCopy_CollectSizeDb",int, float))
608     dbSizeFin = origine.dbsize()
609 
610     assert dbSizeFin   == dbSizeDebut + 10
611 
612 
613 def testCollectRejeux():
614 
615     "test si le rejeu ne modifie le nombre d'élements dans la base"
616 
617     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
618     destination = linda.universe._rd(("destination",linda.TupleSpace))[1]
619 
620 
621 
622     # creation sur le tupleSpace origine
623     for element in xrange(0, 10):
624         aleaOne = random.randint(1,1000)
625         aleaTwo = random.random()
626         origine._out(("testCollectRejeux",aleaOne, aleaTwo))
627 
628     dbSizeDebut = origine.dbsize()
629 
630     nb = origine.collect(destination,("testCollectRejeux",int, float))
631 
632     # rejeux
633     nb = origine.collect(destination,("testCollectRejeux",int, float))
```

```
634
635
636     dbSizeFin = origine.dbsize()
637
638     assert dbSizeFin == dbSizeDebut
639
640
641
642 def testCollectConsommation():
643     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
644     destination = linda.universe._rd(("destination",linda.TupleSpace))[1]
645
646     # creation sur le tupleSpace origine
647     for element in xrange(0, 10):
648         aleaOne = random.randint(1,1000)
649         aleaTwo = random.random()
650         origine._out(("testCollectConsommation",aleaOne, aleaTwo))
651
652     nb = origine.collect(destination,("testCollectConsommation",int, float))
653
654     tupleVide = origine._rdp(("testCollectConsommation",int, float))
655
656     assert tupleVide == ()
657
658
659 def testCollectDestination():
660     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
661     destinationColDest = linda.TupleSpace()
662
663     # creation sur le tupleSpace origine
664     for element in xrange(0, 10):
665         aleaOne = random.randint(1,1000)
666         aleaTwo = random.random()
667         origine._out(("testCollectDestination",aleaOne, aleaTwo))
668
669     nb = origine.collect(destinationColDest,("testCollectDestination",int, float))
670
671     #on cosomme
672     for compteur in range (0,10):
673         destinationColDest._in(("testCollectDestination",int, float))
674
675     tupleVide = destinationColDest._rdp(("testCollectDestination",int, float))
676
677     assert tupleVide == ()
678
679
680 def TestEval():
681     pass
682
683
684 def teardown_func():
685     """
686     PostCondition
687
688     """
689     pass
690
691
692
```

693

Typographical Conventions for Python

Python:Normal Text	**Python:Definition Keyword**	Python:Operator
Python:String Substitution	**Python:Command Keyword**	**Python:Flow Control Keyword**
Python:Builtin Function	Python:Special Variable	Python:Extensions
Python:Exceptions	**Python:Overloaders**	Python:Preprocessor
Python:String Char	Python:Long	Python:Float
Python:Int	Python:Hex	Python:Octal
Python:Complex	*Python:Comment*	Python:String
Python:Raw String	**Alerts:Normal Text**	**Alerts:Alert**

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-


__author__ = "Charles Duysinx aka MonsieurCloug"
__copyright__ = "Copyright 2009, Charles Duysinx"
__license__ = "MIT"
__version__ = "0.0"
__revision__ = "$LastChangedRevision: 175 $"[22:-2]
__date__ = "$LastChangedDate: 2009-03-17 16:15:55 +0100 \
           (Mar, 17 Mar 2009) $"[18:-2]



import sys
sys.path.append('/home/gdh2/Documents/memoire/workspace/redlindalock/src/')
import kernel as linda
#import linda
from nose.tools import *
import random
import py
from kernel import *



"""voici les variables membres à modifier pour que cela
aille plus vite.

# temps imparti pour mettre un verrou
self.delaiVerrouTransactionServeur = 1

# temps imparti pour mettre un verrou
self.delaiVerrouTsServeur = 10

# temps imparti pour mettre un verrou
self.delaiVerrouGlobalServeur = 10

# durée d'une requete non bloquante.
self.delaiUnblockRequete = 1

"""


def testRedLindaError():
    ## non nécessaire
    pass


```

```
49
50 def testRedBlockError():
51     # dépassement de délai pour une requete blocante
52
53     origine=linda.TupleSpace()
54     linda.universe._out(("origine",origine))
55     origine=None
56
57     origine = linda.universe._rd(("origine",linda.TupleSpace))[1]
58     print "ofuoqiufoiqsdufiouo",origine
59
60     origine.delaiBlockRequete = 0   # durée d'une requete bloquante
61
62     py.test.raises(RedBlockError, \
63                    "origine._in(('testRedBlockError',1))")
64
65
66
67 def testRedLindaErrorNotImplemented():
68
69     notImplemented=linda.TupleSpace()
70     notImplemented._out(("notImplemented",12))
71
72     # pour le le cas 2
73     notImplemented.gestionTransaction = 2
74
75     py.test.raises(RedLindaErrorNotImplemented, \
76                         'notImplemented._rd(("notImplemented",12))')
77
78
79     # pour le le cas 3
80     notImplemented.gestionTransaction = 3
81     py.test.raises(RedLindaErrorNotImplemented, \
82                         'notImplemented._rd(("notImplemented",12))')
83
84
85
86 def testRedLockTransactionErrorAcquisition():
87     origine=None
88     origine = linda.TupleSpace()
89     origine._out(("testRedLockTransactionErrorAcquisition",1))
90
91     # pour aller plus vite
92     origine.delaiBlockRequete = 0   # durée d'une requete bloquante
93
94     # je prends le jeton transaction
95     result=origine.set("red:lock:tr", \
96                         str(redLindaAgentId), preserve=True)
97
98     # je vérifie que je l'ai bien
99
100    assert result==1
101
102    py.test.raises(RedLockTransactionError,\
103        'origine._rd(("testRedLockTransactionErrorAcquisition",1))')
104
105    # je le supprime
106    result=origine.delete("red:lock:tr")
107
```

```
108
109 def testRedLockTransactionErrorRelaxe():
110     origine=None
111     origine=linda.TupleSpace()
112
113     # pour aller plus vite
114     origine.delaiBlockRequete = 0   # durée d'une requete bloquante
115     # j'efface tous les jetons qui pourraient trainer
116     result=1
117     while result:
118         print "voici le carigloug", result
119         result=origine.delete("red:lock:tr")
120         print "voici le carigloug", result
121
122     py.test.raises( RedrelaxeLockTransactionError,\
123                     'origine.deVerrouillageTransaction()')
124
125
126 def testRedLockUseTsError():
127     # not implemented
128     pass
129
130
131 def RedLockRedisError():
132     # not implemented
133     pass
134
135
136
```

Typographical Conventions for Python

Python:Normal Text	**Python:Definition Keyword**	Python:Operator
Python:String Substitution	**Python:Command Keyword**	**Python:Flow Control Keyword**
Python:Builtin Function	Python:Special Variable	Python:Extensions
Python:Exceptions	**Python:Overloaders**	Python:Preprocessor
Python:String Char	Python:Long	Python:Float
Python:Int	Python:Hex	Python:Octal
Python:Complex	*Python:Comment*	Python:String
Python:Raw String	**Alerts:Normal Text**	**Alerts:Alert**

```
 1 from numpy import fft,array,arange,zeros,dot,transpose
 2 from math import sqrt,cos,pi
 3 from numpy import fft,array,arange,zeros,dot,transpose
 4 from math import sqrt,cos,pi
 5 #"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""3
 6
 7
 8
 9 qmatrix = [[ 16 , 11 , 10 , 16 , 24  , 40  , 51  , 61 ],
10           [ 12 , 12 , 14 , 19 , 26  , 58  , 60  , 55 ],
11           [ 14 , 13 , 16 , 24 , 40  , 57  , 69  , 56 ],
12           [ 14 , 17 , 22 , 29 , 51  , 87  , 80  , 62 ],
13           [ 18 , 22 , 37 , 56 , 68  , 109 , 103 , 77 ],
14           [ 24 , 35 , 55 , 64 , 81  , 104 , 113 , 92 ],
15           [ 49 , 64 , 78 , 87 , 103 , 121 , 120 , 101],
16           [ 72 , 92 , 95 , 98 , 112 , 100 , 103 , 99 ] ]
17
18 class DCT(object):
19     def __init__(self):
20         pass
21
22     def __transKernel(self,N):
23         A = zeros((N,N))
24         for x in xrange(0,N):
25             for u in xrange(0,N):
26                 if u==0:
27                     A[x][u] = sqrt(1/float(N))
28                 else:
29                     A[x][u] = sqrt(2/float(N))*cos(pi*(2*x+1)*u/float(2*N))
30         return A
31
32     def __itransKernel(self,N):
33         A = zeros((N,N))
34         for x in xrange(0,N):
35             for u in xrange(0,N):
36                 if x==0:
37                     A[x][u] = sqrt(1/float(N))
38                 else:
39                     A[x][u] = sqrt(2/float(N))*cos(pi*(2*u+1)*x/float(2*N))
40         return A
41
42     def transform(self,m):
43         tk = self.__transKernel(len(m))
44         t1 = dot(m,tk)
45         t1 = transpose(t1)
46         t1 = dot(t1,tk)
47         return transpose(t1)
```

```
48
49      def itransform(self,m):
50          tk = self.__itransKernel(len(m))
51          t1 = dot(m,tk)
52          t1 = transpose(t1)
53          t1 = dot(t1,tk)
54          return transpose(t1)
55
56      dct_test = [ [-76 , -73 , -67 , -62 , -58 , -67 , -64 , -55 ],
57                   [ -65 , -69 , -73 , -38 , -19 , -43 , -59 , -56 ],
58                   [-66 , -69 , -60 , -15 ,  16 , -24 , -62 , -55 ],
59                   [-65 , -70 , -57 , -6  ,  26 , -22 , -58 , -59 ],
60                   [-61 , -67 , -60 , -24 ,  -2 , -40 , -60 , -58 ],
61                   [-49 , -63 , -68 , -58 , -51 , -60 , -70 , -53 ],
62                   [-43 , -57 , -64 , -69 , -73 , -67 , -63 , -45 ],
63                   [-41 , -49 , -59 , -60 , -63 , -52 , -50 , -34 ] ]
64
65
66      idct_test = [ [-416 , -33 , -60 ,  32 ,  48 , -40 , 0 , 0 ],
67                    [   0 , -24 , -56 ,  19 ,  26 ,   0 , 0 , 0 ],
68                    [ -42 ,  13 ,  80 , -24 , -40 ,   0 , 0 , 0 ],
69                    [ -56 ,  17 ,  44 , -29 ,   0 ,   0 , 0 , 0 ],
70                    [  18 ,   0 ,   0 ,   0 ,   0 ,   0 , 0 , 0 ],
71                    [   0 ,   0 ,   0 ,   0 ,   0 ,   0 , 0 , 0 ],
72                    [   0 ,   0 ,   0 ,   0 ,   0 ,   0 , 0 , 0 ],
73                    [   0 ,   0 ,   0 ,   0 ,   0 ,   0 , 0 , 0 ] ]
74
75      if __name__=="__main__":
76          dct = DCT(2)
77          print "testing transform..."
78          print dct.transform(dct_test)
79          print "testing itransform..."
80          print dct.itransform(idct_test)
81
```

Typographical Conventions for Python

Python:Normal Text	**Python:Definition Keyword**	Python:Operator
Python:String Substitution	**Python:Command Keyword**	**Python:Flow Control Keyword**
Python:Builtin Function	Python:Special Variable	Python:Extensions
Python:Exceptions	**Python:Overloaders**	Python:Preprocessor
Python:String Char	Python:Long	Python:Float
Python:Int	Python:Hex	Python:Octal
Python:Complex	*Python:Comment*	Python:String
Python:Raw String	**Alerts:Normal Text**	**Alerts:Alert**

```
1  # -*- coding: utf-8
2
3
4  ########################################################
5  # Attention veuillez mettre le path du module redlinda
6  ########################################################
7  #extension du path
8  import sys
9  sys.path.append('/home/gdh2/Documents/memoire/code/linda/')
10
11 #import de la version de production
12 import redlindalock as linda
13
14 # import de la version test
15 import redlindalock as linda
16 #import linda
17
18 # pour le test
19 from cPickle import dumps, loads
20 import time
21
22 if __name__ == '__main__':
23
24     linda.connect()
25     ## master
26
27     monTs=linda.TupleSpace()
28     linda.universe._out(("monTs", monTs))
29     print monTs
30     for elem in xrange(0,10):
31         print elem
32         monTs._out(("somme",elem,elem*2))
33
34     ## master
35     print "j'attends le travail"
36
37     for elem in xrange(0,10):
38         a=monTs._in(("resultat",int))
39         print "voilà le travail", a
```

Typographical Conventions for Python

Python:Normal Text	**Python:Definition Keyword**	Python:Operator
Python:String Substitution	**Python:Command Keyword**	**Python:Flow Control Keyword**
Python:Builtin Function	Python:Special Variable	Python:Extensions
Python:Exceptions	**Python:Overloaders**	Python:Preprocessor
Python:String Char	Python:Long	Python:Float
Python:Int	Python:Hex	Python:Octal
Python:Complex	*Python:Comment*	Python:String
Python:Raw String	**Alerts:Normal Text**	**Alerts:Alert**

```
1  # -*- coding: utf-8
2
3  ######################################################
4  # Attention veuillez mettre le path du module redlinda
5  ######################################################
6  #extension du path
7  import sys
8  sys.path.append('/home/gdh2/Documents/memoire/code/linda/')
9
10 #import de la version de production
11 import redlindalock as linda
12
13 # import de la version test
14 import redlindalock as linda
15 #import linda
16
17 # pour le test
18 from cPickle import dumps, loads
19 import time
20
21 #from redlindalock.kernel import connect, universe, uts, TupleSpace
22
23 if __name__ == '__main__':
24
25     linda.connect()
26     monTs = linda.universe._rd(("monTs", linda.TupleSpace))[1]
27
28     for elem in xrange(0,10):
29         a=monTs._in(("somme",int,int))
30         somme=a[1]+a[2]
31         print "voici la somme de "+ str(a[1])+" + " +str(a[2]), somme
32         monTs._out(("resultat",somme))
```

Typographical Conventions for Python

Python:Normal Text	**Python:Definition Keyword**	Python:Operator
Python:String Substitution	**Python:Command Keyword**	**Python:Flow Control Keyword**
Python:Builtin Function	Python:Special Variable	Python:Extensions
Python:Exceptions	**Python:Overloaders**	Python:Preprocessor
Python:String Char	Python:Long	Python:Float
Python:Int	Python:Hex	Python:Octal
Python:Complex	*Python:Comment*	Python:String
Python:Raw String	**Alerts:Normal Text**	**Alerts:Alert**

```
# -*- coding: utf-8 -*-
import Image
import sys
"""Pour la transformation d'une image en rgb j'utilise la norme
ITU-R 601-2 luma transform:
        L = R * 299/1000 + G * 587/1000 + B * 114/1000
"""

class Color2black():


    def __init__(self):
        self.RedLuma=299.0/1000
        self.GreenLuma=587.0/1000
        self.BlueLuma=114.0/1000
        self.suffixImgBlack="_bnItuR6012"
        self.dir2SaveDefault="bnItuR6012"



    def charge(self,path2img):
        """Pour le moment je gère le nom pour la sauvegarde
        mais il faudra bien évidement faire cela dans une autre
        partie du code
        de plus il ne vérifie pas si le répertoire existe donc

        """

        self.path2img=path2img
        self.blackData=[]
        self.img=Image.open(path2img)
        self.largeurImg=self.img.size[0]
        self.hauteurImg=self.img.size[1]
        self.nbPixel=self.hauteurImg*self.largeurImg

        ## définition noms par defaut des fichiers et des répertoires


        ##le nom avec extension
        b=self.path2img.split("/")
        nomPlusExtension=b[-1].split(".")

        self.path2save="./" \
                       + self.dir2SaveDefault  \
                       + "/" \
                       + nomPlusExtension[0] \
                       + self.suffixImgBlack \
                       + "." +nomPlusExtension[-1]
```

```
49
50      def toBlack(self):
51          """ Cette procédure ne vérfifie pas si cette opération est
52          possible.
53          """
54
55          self.dataImg=list(self.img.getdata())
56          r, g, b = self.img.split()
57          self.canalRouge=list(r.getdata())
58          self.canalVert=list(g.getdata())
59          self.canalBleu=list(b.getdata())
60
61
62          for s in range (self.nbPixel):
63              pixel=int((self.canalRouge[s] * self.RedLuma) \
64                    + (self.canalVert[s] * self.GreenLuma) \
65                  + (self.canalBleu[s] * self.BlueLuma))
66
67              self.blackData.append(pixel)
68
69      def saveImg(self):
70          self.blackImg=Image.new("L",self.img.size,100)
71          self.blackImg.putdata(self.blackData)
72
73          self.blackImg.save(self.path2save)
74
75      def affiche(self):
76          self.blackImg.show()
77
78  if __name__=='__main__':
79      monImage=Color2black()
80      monImage.charge("./img_in/cabane.jpg")
81      monImage.toBlack()
82      monImage.saveImg()
83      monImage.affiche()
84
```

Typographical Conventions for Python

Python:Normal Text	**Python:Definition Keyword**	Python:Operator
Python:String Substitution	**Python:Command Keyword**	**Python:Flow Control Keyword**
Python:Builtin Function	Python:Special Variable	Python:Extensions
Python:Exceptions	**Python:Overloaders**	Python:Preprocessor
Python:String Char	Python:Long	Python:Float
Python:Int	Python:Hex	Python:Octal
Python:Complex	*Python:Comment*	Python:String
Python:Raw String	**Alerts:Normal Text**	**Alerts:Alert**

```
1  # -*- coding: utf-8
2
3  ########################################################
4  # Attention veuillez mettre le path du module redlinda
5  ########################################################
6  #extension du path
7  import sys
8  sys.path.append('/home/gdh2/Documents/memoire/code/linda/')
9
10 #import de la version de production
11 import redlindalock as linda
12
13 # import de la version test
14 import redlindalock as linda
15 #import linda
16
17 # pour le test
18 from cPickle import dumps, loads
19 import time
20
21 class Color2BlackMaster():
22
23     def __init__(self):
24         self.suffixImgBlack="_bnItuR6012"
25         self.dir2SaveDefault="bnItuR6012"
26
27     def saveImg(self):
28         self.blackImg=Image.new("L",self.img.size,100)
29         self.blackImg.putdata(self.blackData)
30         self.blackImg.save(self.path2save)
31
32     def affiche(self):
33         self.blackImg.show()
34
35     def toBlack(self, ts):
36         """ Cette procédure ne vérfifie pas si cette opération est
37         possible.
38         """
39
40         self.dataImg=list(self.img.getdata())
41         r, g, b = self.img.split()
42         self.canalRouge=list(r.getdata())
43         self.canalVert=list(g.getdata())
44         self.canalBleu=list(b.getdata())
45
46         currentblock = 0
47         currentR = []
48         currentV = []
```

```
49          currentB = []
50
51          for s in range (self.nbPixel):
52            if s%1000 == 999 :
53                ts._out(("toBlack",currentblock,currentR,currentV,currentB))
54                currentblock = currentblock+1
55                currentR = []
56                currentV = []
57                currentB = []
58
59            currentR.append(self.canalRouge[s])
60            currentV.append(self.canalVert[s])
61            currentB.append(self.canalBleu[s])
62
63          if self.nbPixel%1000 <> 999:
64              ts._out(("toBlack",currentblock,currentR,currentV,currentB))
65
66          for i in range (0,currentblock) :
67              tuple = ts._in(("OKBlack", i,list))
68              pixel= tuple[2]
69
70
71              self.blackData.append(pixel)
72
73
74
75      def charge(self,path2img):
76          """Pour le moment je gère le nom pour la sauvegarde
77          mais il faudra bien évidement faire cela dans une autre
78          partie du code
79          de plus il ne vérifie pas si le répertoire existe donc
80          """
81          self.path2img=path2img
82          self.blackData=[]
83          self.img=Image.open(path2img)
84          self.largeurImg=self.img.size[0]
85          self.hauteurImg=self.img.size[1]
86          self.nbPixel=self.hauteurImg*self.largeurImg
87
88          ## définition noms par defaut des fichiers et des répertoires
89
90
91          ##le nom avec extension
92          b=self.path2img.split("/")
93          nomPlusExtension=b[-1].split(".")
94
95          self.path2save="./" \
96                         + self.dir2SaveDefault  \
97                         + "/" \
98                         + nomPlusExtension[0] \
99                         + self.suffixImgBlack \
100                        + "." +nomPlusExtension[-1]
101
102 if __name__ == '__main__':
103
104     linda.connect()
105
106     ## master
107     monTs=linda.TupleSpace()
```

Typographical Conventions for Python

Python:Normal Text	**Python:Definition Keyword**	Python:Operator
Python:String Substitution	**Python:Command Keyword**	**Python:Flow Control Keyword**
Python:Builtin Function	Python:Special Variable	Python:Extensions
Python:Exceptions	**Python:Overloaders**	Python:Preprocessor
Python:String Char	Python:Long	Python:Float
Python:Int	Python:Hex	Python:Octal
Python:Complex	*Python:Comment*	Python:String
Python:Raw String	Alerts:Normal Text	**Alerts:Alert**

```
# -*- coding: utf-8

#extension du path

# -*- coding: utf-8

##########################################################
# Attention veuillez mettre le path du module redlinda
##########################################################
#extension du path
import sys
sys.path.append('/home/gdh2/Documents/memoire/code/linda/')

#import de la version de production
import redlindalock as linda

# import de la version test
import redlindalock as linda
#import linda

# pour le test
from cPickle import dumps, loads
import time

#from redlindalock.kernel import connect, universe, uts, TupleSpace
class Color2BlackAgent():
    def __init__(self):
        self.RedLuma=299.0/1000
        self.GreenLuma=587.0/1000
        self.BlueLuma=114.0/1000

    def toBlack(self, tupleATraiter ):
        tupleRetour = []

        for i in range(tupleATraiter[1]) :
            tupleRetour.append(int ((tupleATraiter[2][i] * self.RedLuma) \
                        +  (tupleATraiter[3][i] * self.GreenLuma) \
                        +  (tupleATraiter[4][i] * self.BlueLuma)))

        return tupleRetour

if __name__ == '__main__':
    linda.connect()
    monTs = linda.universe._rd(("monTs", linda.TupleSpace))[1]
    print "toto"
    agent = Color2BlackAgent()
    currenblock=0
    while monTs._rdp(("toBlack",currenblock,list,list,list)):
```

```
49          a=monTs._in(("toBlack",currenblock,list,list,list))
50          monTs._out(("OKBlack", a[1], agent.toBlack(a)))
51          currenblock=currenblock+1
52
```

```
108     linda.universe._out(("monTs", monTs))
109     print monTs
110
111     monImage=Color2BlackMaster()
112     monImage.charge("./img/cabane.jpg")
113     monImage.toBlack(monTs)
114     monImage.saveImg()
115
116     monImage.affiche()
```